THE UNIVERSITY OF ALBERTA

ALBERTA DISCRETE ACTIVITY MONITOR SYSTEM (ADAMS)

BY

© S. WILLIAM HAYWARD

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF CHEMICAL AND PETROLEUM ENGINEERING

EDMONTON, ALBERTA

FALL, 1972

ALBERTA DISCRETE ACTIVITY MONITOR SYSTEM (ADAMS)

BY

S. WILLIAM HAYWARD

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF CHEMICAL AND PETROLEUM ENGINEERING

THE UNIVERSITY OF ALBERTA

EDMONTON, ALBERTA

FALL, 1972

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "ALBERTA DISCRETE ACTIVITY MONITOR SYSTEM (ADAMS)" submitted by S. William Hayward in partial fulfilment of the requirements for the degree of Master of Science in Chemical Engineering.

ABSTRACT

The Alberta Discrete Activity Monitor System (ADAMS) is a generalized system designed to apply computer control to discrete, real-time operations. Typical applications include plant start-up and shutdown, system checkout, and recurring functions such as those associated with batch operations. ADAMS is designed to assist with each phase of project development from management level planning, through detailed project specification and testing, to regular implementation and optimization.

The distinguishing feature of ADAMS is that it uses the network analysis concepts popularized by critical path methods (CPM) as a basis for project planning and real-time operation. ADAMS accepts a coded version of the same network developed during the project planning stage and uses it in conjunction with "activity latest start times" (calculated by a standard, offline CPM analysis) to control the online scheduling of each activity. This eliminates the need for the user to program the sequence in which activities should be executed, and separates the detailed specification of the activities from the project planning. It also permits ADAMS to initiate activities in accordance with real-time requirements rather than depending on an apriori fixed sequence.

ADAMS contains features which aid the computer system engineer, the applications engineer, and the process operator but the function of each are kept distinctly separate. The computer system engineer is presented with a modular program that separates online

and offline functions. ADAMS is designed to run on real-time process computers in either a timeshared or dedicated environment. It has been written to be independent of any particular application and is relatively independent of the computer hardware and software system on which it is run. The coding used by the application engineer in implementing the functions to be accomplished by each activity is a "fill-in-the-blanks" procedure designed to assist non-computer oriented personnel. He must simply select an appropriate series of generalized routines for each activity from a library of unit functions and specify the required parameters. The process operator is kept fully informed by messages originating within ADAMS or in the user specified activities, and can take over complete or partial control of the operation at any time. Manual and automatic operations can be intermixed and ADAMS can also communicate with and control other application programs such as direct digital control (DDC) systems.

ADAMS has been implemented successfully on an IBM 1800 computer operating under the multi-programming executive (MPX) system. Its usefulness has been demonstrated by application to the start-up of a pilot plant evaporator, including transfer to DDC control for continuous operation.

The present ADAMS programs do not contain all the features in the recommended design but demonstrate the feasibility of a generalized approach to the control of discrete activities and is an excellent basis for the development of a commercial system.

ACKNOWLEDGEMENTS

The author expresses his sincerest thanks to Dr. D. Grant Fisher who devoted much of his time and energies in giving appreciated and needed advice concerning this work.

Many thanks are also expressed to the staff of the Data Acquisition, Control, and Simulation Centre and in general the staff and graduate students in the Department of Chemical and Petroleum Engineering at the University of Alberta who made the duration of the work a pleasure.

The work was generously financed by the International Business Machines Corporation and the Department of Chemical and Petroleum Engineering of the University of Alberta.

Above all, the author thanks his parents for their encouragement and the opportunity that they made possible for him to carry out these studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Process Control by Digital Computers

The continued growth of computer control will depend to a considerable extent on the availability of high level, application-oriented, software systems. Such systems permit the user to rapidly implement computer applications with minimum assistance from the vendor and/or programming specialists. Direct digital control (DDC) programs[1] and supervisory control systems such as PROSPRO[2] and BICEPS[3] are excellent examples of application programs which have greatly assisted computer control of continuous processes. However, to date, there has been relatively little effort directed toward the generalized control of plant functions which consist of single or repeated sequences of discrete steps such as are found in batch operations, start-up, system checkout, laboratory applications, etc.

Development in this area has not lagged because of a lack of potential applications. It has lagged because of the difficulty in developing a generalized approach. Each application involves thousands of separate steps that must be programmed. For example, with DDC a few standard control algorithms can be applied, through the appropriate selection of parameters, to the majority of industrial control problems. In contrast, the start-up of each piece of equipment requires a different sequence of operations and checks. Thus computer control of discrete activities is more difficult but not impossible.

A very high degree of flexibility is required within such a system because of the nature of automation. Normally, the first step

would entail automating the existing operating procedures. Future work would no doubt involve reworking such procedures to take greater advantage of computer capabilities. It is necessary to permit the initial work plus following changes to be implemented gradually with a minimum upset of plant operations.

1.2 Objectives of the Study

The objective of the study is the application of computer control to process operations consisting of discrete steps such as plant start-up, shutdown, batch processes or system checkout.

Planning for the system proceeded on the basis that any particular application would be handled in five distinct phases (see Figure 1.1):

1. project planning and organization
2. programming details of project for computer
3. testing
4. implementation
5. project re-evaluation and improvement

The study is directed at aiding work in each of these five phases to give the prospective user a definite framework or plan for adopting computer control of discrete operations.

The first specific objective is to make the program as independent as possible of the application it controls and of the computer software and hardware system on which it is to be run. Secondly, the system is to be designed to assist in developing the initial organization of the project as well as handle as many of the complex scheduling tasks during execution as possible. These contributions would greatly

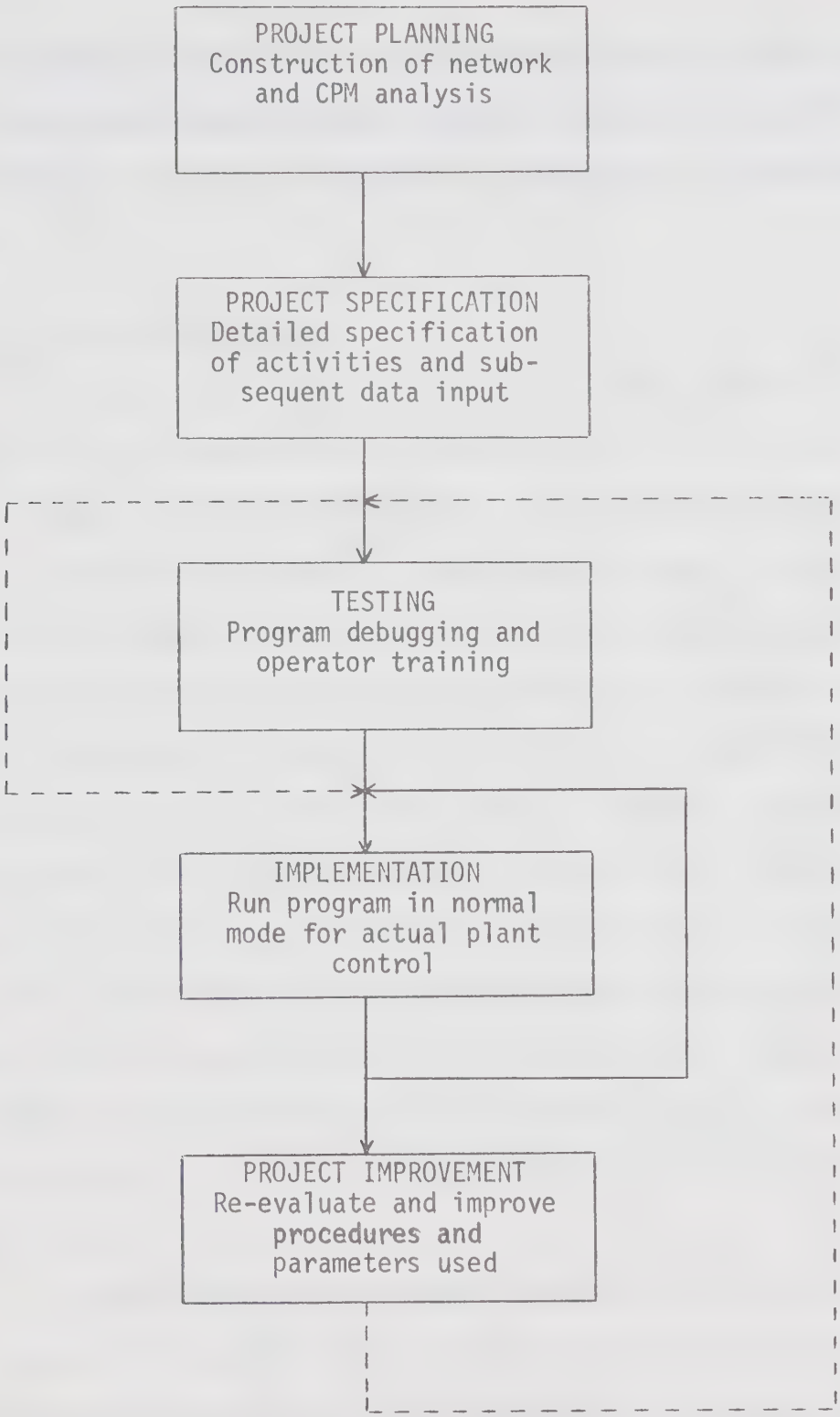


Figure 1.1. MAJOR STEPS IN DEVELOPING A NEW APPLICATION

reduce the amount of supervision and bookkeeping left up to the user. Finally, the system should be user oriented in order to reduce the amount of computer expertise, especially in programming, required of the user.

1.3 Basic Approach of Project

The first major decision was to use network analysis[4] in the planning of each application. This widely used tool provides an excellent managerial/technical interface. A project is defined in terms of a network of activities. These activities may in turn be subdivided into networks consisting of more detailed activities; the activities on the most detailed level of network, for this use, being composed of programmable steps. The main advantage of defining a job on various levels is that work on one level of network is independent of work on more or less detailed levels. Work on higher levels only affects the initiation time of activities on lower levels. Work on lower levels only defines the method employed within activities on that particular level and usually does not affect the higher levels. In other words, the functions and sequencing dependencies of activities within a level can be planned or changed with minimum or no adjustments on other levels.

The next decision was to adopt the network concept as the basis to control the sequencing of activities during execution. This allows the user to use his most detailed network as input data without additional instructions to the system concerning the order of sequencing. The network approach also provides a convenient method of documenting the project and displaying progress during actual execution. The

network arrow diagram with present position of execution can easily be displayed on a computer driven plotter or scope for the operator's information.

Critical path analysis[4], a network analysis technique, is used to obtain activity latest start times (LST) to be used as priorities in sequencing and executing the activities in order to achieve a time optimal project. Resource allocation policies may also be implemented in the same manner.

In addition to controlling the initiation of new activities, it was decided to design the real-time executive to supervise the execution of any number of activities which could occur concurrently and to accommodate real-time factors that affect scheduling. For example, when several activities are ready to be initiated and/or executed the activity latest start times give a relative idea of which are more critical. Also, should a particular activity or sequence of activities fall behind schedule because of wrong apriori estimates of the time required or because of real-time setbacks (eg. equipment failure) its priority will become the highest. Hence an ordinarily non-critical (timing) activity could be given top priority if real-time events delayed its initiation. A priority system for initiating new activities is important considering the number of parallel occuring activities will normally be restricted by core limitations, which in some cases will necessitate the queuing of low priority activities until core space becomes available.

Another decision was to use a "table driven processor" approach to implement a high level user language to be used in programming the activities. Basically a set of subroutines was developed to

perform the desired basic functions and the user simply identifies which subroutine he wishes to execute and supplies the required parameters. The set contains the basic arithmetic and logic capabilities for control calculations and decisions, but also has high level functions for process I/O, operator communications, and high speed monitoring functions. Programming is a "fill in the blanks" procedure that is easily learned without prior computer programming experience.

1.4 Previous Work

As previously explained, work in the field of computer control of sequential or discrete type operations is not as advanced as that for continuous operations. This is mainly because the complexity arising from the multitude of details makes the organization and programming very difficult. Papers on the subject, although very few, are generally in agreement about the existing difficulties but conclude that computer control is feasible and can be economical. The need for software development is emphasized. Some of the more general comments are expressed below.

Itahara[6] claims the justification for computer control of batch processes lies in labour reduction and improved productivity. He emphasizes the difference between batch and continuous operations, stresses the importance of scheduling equipment to maximize usage, and states that a major problem is the development of a scheduling algorithm to achieve efficient operation. The computer's high speed and ability to swap programs are necessities when controlling several processes or more than one step in a single process. An extremely important point made is that the development of software can often

result in costs that surpass those of initial machine investment. Other major problems are the need for equipment/computer failure diagnostic programs, the development of a smoothly working system and testing the system before using it.

Bacher and Kaufman[7], also on the subject of computer controlled batch chemical reactions, developed a prototype computer system which they state improved product reproducibility and quality.

Carlo-Stella[8] points out that "sequential logic is usually highly customized to the particular applications". This confirms the view that development of a generalized program to control such operations is both needed and difficult.

In an article on automatic plant start-up using computers, published in 1967, Baldridge[9] includes a literature survey which concludes" practically nothing has been done to apply computers for the automatic start-up of chemical processes". Briefly his incentives for computer control include:

1) errors in human judgement may lead to expensive losses or hazardous conditions during start-up;
2) design and operating cost savings due to tighter control of transient stages;
3) more time available for steady state operation;
4) extra labour normally required during start-up can be reduced;
5) feasibility of processes now considered "too lively" to be handled by operators during start-up or shutdown.

The work reported by Baldridge included the programming of a start-up

routine which was tested on a pilot plant. Direct digital control was used, a high level language developed and problems such as hot and cold start as well as emergency and normal shutdowns were considered. Little about the program itself was described.

Whitman[10] in discussing start-up of a pump under computer control underscores the large number of details involved in such superficially simple operations.

Baker[11] in a thorough analysis of digital control of batch processes lists significant differences between batch and continuous operations which make batch automation very difficult but also makes the digital control system a potentially powerful tool. He also remarks on the lack of technical knowledge in the field and claims "many commerically available software systems are severely inadequate for batch automation". He concludes that "usually computer control of batch operations has a handsome payout".

Recent papers, describing newly developed high level process control oriented languages[12,13], claim to aid the process engineer in programming sequential control programs. Other programs which have been developed by computer manufacturers[2,3,14,15] exclusively for their own machines have features which enable the project engineer to write his program. Control of sequential processes is included among their potential applications.

Of the systems examined, most placed the burden of sequencing and scheduling in the users' programs. The systems lacked flexibility required for general usage and none had a framework to assist the user in project organization.

1.5 Structure of the Thesis

The basis for the system design is outlined in Chapters II and III and a prototype of the system was implemented on an IBM 1800 Data Acquisition and Control System in order to test and evaluate various design alternatives. The system (ADAMS - Alberta Discrete Activity Monitoring System) is written in FORTRAN and works in a time-sharing environment under the standard IBM MPX executive[5]. ADAMS can use either the DDC program which has been functional at the University of Alberta since 1968 or a user supplied program for process I/O. Practical use of ADAMS obtained by starting a pilot plant double effect evaporator is outlined in Chapter V.

There is some redundancy of material in each chapter to make them relatively independent of each other so that the reader can refer directly to his area of interest.

In conjunction with the thesis, a program user's manual [17] contains detailed information on ADAMS.

CHAPTER II

DESIGN FEATURES CONSIDERED ESSENTIAL FOR GENERAL PROCESS CONTROL PROGRAMS

2.1 Introduction

This chapter is written to outline general design features which are considered essential for a good process control program. These programs must be readily adaptable to any process; they must be general with respect to computing and plant equipment, plus be designed for use by non-computer oriented staff. The features presented here are for any program which monitors and/or controls a real-time process, not necessarily for discrete processes which are the subject of the rest of the thesis. The discussion has been summarized from the viewpoints of:

1) the project engineering team who are not necessarily familiar with computer techniques but are familiar with what features must be implemented to ensure safe, efficient plant operation.

2) the system engineers who are perhaps unfamiliar with process requirements but who must adapt the program to meet the requirements of a particular application and to run efficiently on their real-time computer.

Any programming system should be implemented so that these functions or interests are kept separate and appear transparent to the other.

2.2 Features Provided for the Project Engineer

This section describes user oriented features considered important in a real-time program. In general the project engineer

should not require an extensive knowledge of computer techniques but will require a thorough knowledge of the plant operation, control techniques, and process calculations as well as the basic capabilities of the program he will use.

The general program should be adaptable to a broad range of computer controlled processes, i.e. it must be adaptable to applications which vary considerably with respect to size, complexity, time scale, automatic vs. manual operations, etc. This means that data required for plant control must be available whether entered by the operator or entered directly from an instrument. Reading data in some cases will be required at high frequency, in other cases only a single piece of information will be required. Both options must be available to the user.

Good communications must be maintained between the operator and the control program. The extent of the communications will depend upon the function of the program but, in general, any error conditions or significant changes in plant position should be relayed to the operator automatically. Other information should be available upon operator request, eg. variables monitored by the program and present status in program.

Operator control over the program is an essential feature. The range of such control could extend from operator approval of all steps taken by the program, for instance in a testing mode, to minimum documentation informing the operator of normal progress. The operator should be able to interrupt or suspend the program at any time and then re-initiate it when desired. In doing this the plant should

not be left uncontrolled or in an unsafe condition.

Control programs should handle real-time problems rather than function solely on the basis of apriori decisions and sequences. The programs could simply transfer control to the operator whenever an unexpected problem arose or could handle the problem automatically.

Every effort should be made to assist the project engineer in utilizing the program. Development of a high level language oriented to process control is considered essential. Terms and operations which are familiar to the project engineer should be used. Testing facilities for online program debugging and other changes should be provided especially in cases requiring extensive input or complicated sequences of operations.

2.3 Features Provided for the Computer System Engineer

The cost of developing software support for individual applications has dictated a need for generalized programs which are readily adaptable to accommodate various types and sizes of plant operations, various computer operating systems and various types of software support. In addition, they must be easily used by a project engineer with little or no computer experience.

Techniques that can be used to aid system generality include modular construction, distinct easily changed software interfaces between the program and external hardware or software, and the use of widely supported computer languages. In some cases these techniques will be less efficient than ones employed in a system designed for one particular application but the loss in efficiency is offset by a gain in generality.

By programming distinct functions into individual modules the various capabilities of the program become clearly defined. Should it be required to add or modify modules the task is then better defined and the work is concentrated on a particular area of the system rather than requiring changes throughout the system. Modules are particularly helpful when making changes to program size and configuration.

Software interfaces are merely small modules which handle I/O between the application program and the system hardware or system support programs (see Figure 2.1). Such interfaces facilitate changes when implementing the program on different computer systems or when changes are made to the computer system on which it is presently running.

Normally, high level languages (eg. FORTRAN compiler) are supported by more than one type of computer and hence programs written in such languages can be used more widely. As mentioned, such languages are inefficient when compared to assembler coding, however, efficiency has been sacrificed for generality.

To assist the user in programming the supplied system a method of coding should be made available in terms which are familiar or easily understood by him. The coding should incorporate symbolic referencing where possible, eg. symbolic references to I/O device addresses. Error diagnostics are essential and "self-documenting" coding reduces the work greatly.

In some cases it will be necessary to replace hardware or system executive functions available through manufacturers by less efficient but more general programming. This is especially apparent in

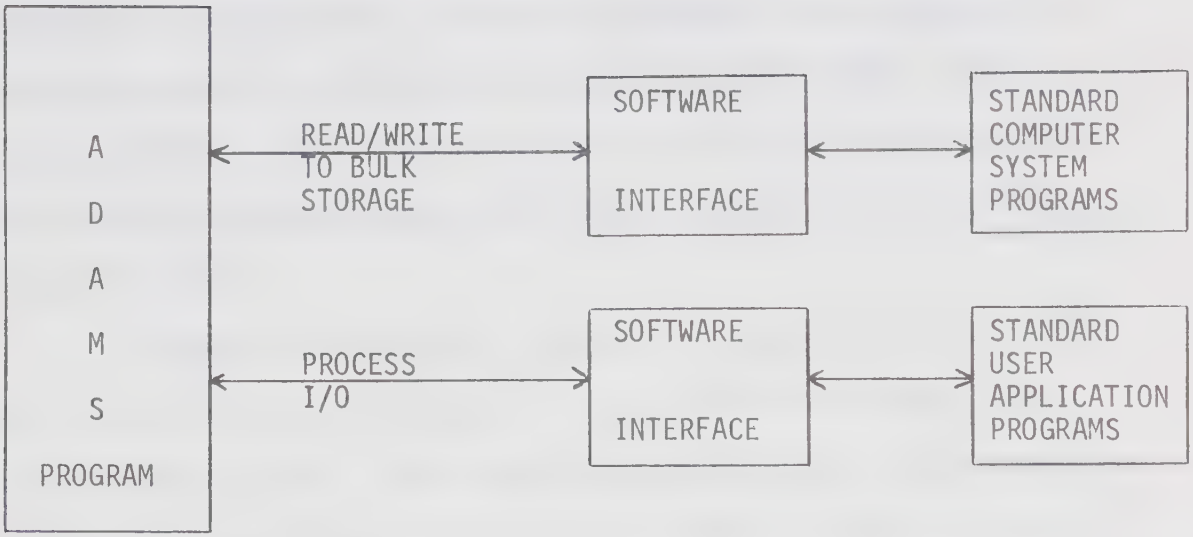


Figure 2.1. USE OF SOFTWARE INTERFACES

the field of operator communications which can tie up the CPU with relatively slow operation. On specific systems I/O is made more efficient by use of buffering or asynchronous operations. This is highly dependent upon the computer and to rely on the technique used by a particular manufacturer would place a severe limit on attaining a general system.

Another key feature to consider is the provision for future system expansion and development. All possible requirements should be looked at and even if not provided immediately the system should be structured to facilitate later addition. Table oriented data areas which are accessible to many programs and are readily expanded is one such technique that may be used.

In programming a process control system, particular attention should be paid to execution time and efficiency of core usage versus the type of program. Programs can be categorized as online or offline and repetitive or once thru. A repetitive online program should be more carefully designed with respect to execution time and size whereas an offline program run infrequently, eg. data input, would not have to be as carefully controlled.

Another consideration in system design is the response time of a program, the main difference being whether the program is stored in bulk storage or is core resident. For control of "fast" processes response time is critical whereas this is not the case if the time base of the process is hours. Relatively fast responses should be given to operator requests so that he may proceed quickly about his job. On the other hand, should the program request operator information,

it should be at the operator's convenience to enter the response not at the demand of the computer. This is keeping within the concept that the computer is a tool and only represents part of the operator's job.

A process control system should recognize that changes during the maintenance and optimization phase of a project are essential. All possible considerations should be made to aid these changes rather than making them an undesirable facet of operation.

CHAPTER III

PHILOSOPHIES, OBJECTIVES, AND POSSIBLE ALTERNATIVES IN DESIGNING A DISCRETE ACTIVITY MONITOR PROGRAM

3.1 Introduction

This chapter considers the general specifications and features of a system to implement computer control of discrete processes such as plant start-up, batch operations, etc. Problems of this nature are characterized by thousands of steps which must be coordinated into a dynamic real-time project. The basic objectives of the design plus the alternatives which were evaluated are presented here.

The discussion in this chapter is organized into the same five general phases followed in most design projects (see Figure 1.1):

1) overall organization
2) detailed planning
3) testing
4) implementation
5) project maintenance and improvement

Each of these topics is dealt with in the following sections.

3.2 Overall Organization of the Project

In general, most projects are organized and planned on several levels. The highest, most general level is usually a management responsibility; the intermediate level involves more detailed design and specification by engineers; and the most detailed level can frequently be implemented by technicians. It is important that any organizational scheme facilitate and coordinate action at all these levels. It is not

desirable to have one approach for project management and another as a basis for computer application.

In large discrete operations, there are always several independent steps which can be done in parallel instead of sequentially. It is important that the organization show where such conditions exist so that advantage may be taken of them to facilitate and speed execution.

The method chosen for organizing the application must also clearly show constraints such as in sequencing (ordering), resource allocation, and target dates. When choices within the constraints may be made, the best choice would result in a time optimal project duration.

Network analysis, a widely accepted management tool with broad applications in handling projects composed of many levels and many details has been adopted for project organization.

The volume or amount of detail is the principal factor which leads to difficulties in organization and coordination of the many levels of planning. Several levels can easily be seen in one application, eg. start entire plant ≠ start plant feed system ≠ start pump ≠ close switch.

The levels within an application must be clearly defined so that steps or decisions made on one level are independent of any other levels. For instance, the time at which a feed system is to be started is decided relative to other steps in starting the plant. However, on a more detailed level, the procedure for starting a pump is not dependent on when the feed system is started. Thus a person planning the start-up of a plant needs only to decide that the feed system has

to be started and when; he does not have to know the mechanics of starting a pump. These details may be assigned to a different level, i.e. person.

The arrow diagram[4], a graphical representation of the project as a network of activities can show a true picture of the steps involved at any level. It provides a flexible format and is realistic in representing any concurrent paths within a project. Any constraints on the project such as sequencing, resource limitations or target dates may be clearly shown.

Network analysis includes procedures for analyzing and evaluating a project. Prominent among these are CPM[4], PERT[4], and RAMPS[4] routines.

It is concluded in this work that network analysis techniques can and should be adopted as the basis for computer control of discrete events.

3.3 Detailed Planning - Programming

After the overall application has been represented on a broad scale as a network, in terms familiar to management, it is necessary to subdivide it into smaller steps or instructions until it is recognizable as individual steps that can be programmed for execution by a computer. This maintains the system concept of breaking any large project into subsystems until each subsystem is capable of being handled as a single entity.

It is an objective of this project that the specification of the individual steps does not require any prior programming knowledge

from the process personnel who have planned the application to this point. Hence the programming, i.e. implementation of the planning, must employ techniques which are oriented toward the computer layman.

Even at relatively high levels of programming language the user is still faced with programming thousands of individual steps. In order to further assist him, the use of macro level instructions and symbolic referencing is recommended. In his application there will be several strings of steps which are identical in function but specific in where they are applied. For example, turn on power to pump motor, wait, check for normal operation, proceed or try restart is the common procedure followed in starting most pumps. The user should have the ability to combine such recurring steps into a macro instruction or its equivalent with the necessary parameters to make it general.

The user should also be able to add instructions specific to his application to the standard instruction set provided with the system.

The ability to alter the program steps quickly and easily is essential. This should not require a complete program reload but be accomplished by specifying a particular instruction in a particular activity to be changed. Provision should also be made to change a parameter or a set of parameters, for example, a desired set of steady state operating conditions, with a single table entry rather than having to change the parameters at every point in the program that they are used.

Four different approaches to producing a user oriented

computer language have been investigated. They are the assembler, higher level languages such as FORTRAN which are compiled, the interpreter, and the table driven processor. Briefly, some of the advantages and disadvantages of each are:

1) assembler - instructions are converted into machine language on essentially a one to one basis. Macro capabilities and compile time options greatly extend the power and attractiveness of assemblers.

advantages -assembler programming is highly flexible and can be very efficient with respect to the use of core space and execution time.

disadvantages - assembler coding requires an experienced programmer.

- a separate assembler is required for each different computer, i.e. assembler coding is usually machine dependent.

- programs must be re-assembled after any alteration, i.e. a complete program reload.

2) compiler - instructions are translated from high level statements to several machine language instructions, eg. FORTRAN.

advantages -the high level statements require less programming time and expertise.

- because of the translation procedure the statements of a compiler language can be standardized and thus supported by several computers.

disadvantages - conversion to machine language usually results in lower efficiency than assembler coding.

- alterations in programs require recompilation.

3) interpreter - source code is examined at execution time and translated into an appropriate set of instructions every time it is to be executed.

advantages -since the program remains in the source form it is easily modified and edited online.

disadvantages - often slow since every execution of an instruction requires time for "interpretation" as well as the extra core required for the interpreter program.

4) table driven processor - consists of a set of general routines which have been written in, or compiled into machine language. Routines utilize user specified parameters which are supplied in a standard format or table.

advantages -permits online editing of parameters.
-routines written in a compiler or assembler language are only compiled or assembled once and only exist in core or on disk once.
-data deck is of minimum size since only essential parameters and options are specified.
-new routines easily added.
-high or low level instructions (routines) are possible.

disadvantages - general routines are less efficient (core and execution time) than a specific one.

It was decided to sacrifice efficiency in execution time and core storage for the more user oriented features provided by a high level, easy to learn, easily modified language. A table driven processor was chosen since it is easily implemented and modified itself. Also the use of a table driven processor lends itself to "fill in the blanks" type programming which has already proven successful in the process industries, eg. PROSPRO[2].

3.4 Normal Program Execution

3.4.1 Introduction

Normally, as shown in Figure 1.1, the phase before execution is project testing, but for a better understanding of the requirements

in testing the execution concepts will be discussed first.

The discussion of the objectives and alternatives which follows is grouped into the major areas of program development.

3.4.2 Scheduling of Activities

The problem is to supervise, in real-time, a multi-step project in which the steps are constrained as to sequencing but are sometimes flexible in that several may be executed simultaneously. The sequencing of the activities within a project can be affected by unplanned occurrences. The duration of a project will depend on one sequence of activities, the critical path, which stretches from the start to the end of the project. However, because of the possibility of real-time problems and mistaken apriori estimations of activity duration neither the critical path nor the project duration should be assumed.

The scheduling of the activities is a major problem which should ideally be handled by the monitor program. Besides the constraints on the ordering of activities within the network, there can also be constraints on resources as previously discussed. However, within all constraints there will sometimes be choices as to which among several activities to execute and the program should make these choices with a time optimal project duration as its goal.

If the monitor program can carry out all the activities in accordance with the user's detailed network diagram then the user's programming job is reduced to specifying the steps for individual activities. He does not have to program or arrange activity sequencing. Also the sequencing and activity specification functions are

separated, i.e. he can change the sequence of activities by modifying only the network or he can change the details of any activity alone without affecting the network.

3.4.3 Efficient Execution of Concurrent Activities

It is frequently desirable to have the computer supervise several separate activities at one time since in most projects there are several activities which can be implemented in parallel and therefore reduce the time required to complete the project. However, this raises the problem of how to allocate the CPU time efficiently among the activities. The simplest approach is to allow only one activity to be in progress at a time and the task of arranging the order of the activities would be left to the user. This would result in a very simple program. The more complex program would supervise several activities at once and as mentioned divide CPU time efficiently among them.

The implications to the user of the method chosen can be illustrated by an example of starting three pumps simultaneously. The major steps for starting one pump are:

1) turn on power to pump motor
2) wait
3) check for normal operation
4) proceed or try restart

Identifying the various steps by $S_n(X)$ for step X in starting pump n the various basic sequences to carry out the operation are:

1) "single sequential chain"

$S_1(1)$, $S_1(2)$, $S_1(3)$, $S_1(4)$, $S_2(1)$, $S_2(2)$, $S_2(3)$, $S_2(4)$, $S_3(1)$, $S_3(2)$, $S_3(3)$, $S_3(4)$

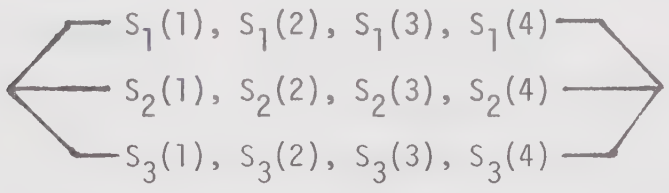
At a minimum, this method will take the sum of the times required to start the three pumps individually and the computer will be idle during steps such as the wait. Also, a delay in starting pump 1 would cause equal setbacks in starting pumps 2 and 3.

2) "single chain with user control of parallel sequencing"

$S_1(1)$, $S_2(1)$, $S_3(1)$, S (2), $S_1(3)$, $S_1(4)$, $S_2(3)$, $S_2(4)$, $S_3(3)$, $S_3(4)$

In this case all pumps are started simultaneously but the checking is done for one pump at a time so a delay in one pump will cause delays in the one(s) following. Note, however, that one wait period serves all three pumps. It would be possible to make the checking parallel but this would require the programmer to set several flags, a task which should be avoided. This approach to programming discrete activities has simplified some existing monitor systems.

3) "three distinct chains working in parallel with monitor controlling the parallel execution"



The monitor will control execution. If it hits a wait period, it may go on to another activity (chain of steps). The user specifies the network which defines the relationship between the three activities and the steps required in each activity. Since they are the same steps in this case, he would probably just have to change a few parameters in order to have the procedure developed for pump 1 also apply to pumps 2

and 3. Once again it should be noted that the tasks of sequencing and programming are independent and can thus be modified separately.

The duration of the longest single activity (critical path) is the duration of the project. The monitor required becomes a complex supervisory program but savings in user programming plus overall project duration make it attractive to the user.

3.4.4 Provisions for Unexpected Real-Time Problems

Real-time problems, besides throwing off apriori estimates of the critical path and project duration, may also upset plant operation in interacting applications (eg. if the cooling medium pump to a heat exchanger fails several minutes after it is started then some of the steps following should be "held" rather than implemented).

Real-time problems in plants are normally handled by 1) shutting down the plant, or 2) holding the plant at a safe status until the error is fixed, then proceeding.

When under computer control, the possibilities that exist are:

1) shutdown
   a) the plant, or
   b) the program and continue on manual operation
2) total suspension of the monitor
   a) correct the situation and re-initiate the monitor from the point of suspension, or
   b) correct the situation and direct the monitor where to re-initiate execution

3) partial suspension of monitor

- let the unaffected activities continue, then re-initiate the suspended ones either at the point of suspension or at a new location

Offline plans can be totally disrupted by unexpected occurrences so that special provisions in the monitor will be required to handle them or turn the situation over to the operator to correct manually. Whatever the case, the plant cannot be left in an unsafe condition.

Since the computer can execute many activities faster than a human operator can respond, any scheme to permit operator intervention or partial control must:

1) utilize a system of flags (logic options) which will force the computer to wait at specified points for instructions from the operator, or

2) check each step against a list of permissable operations before implementing it.

Furthermore, the process operator must have the option of changing the flags or list as he wishes rather than having this done exclusively as an offline task.

3.4.5 Operator Communications

The process operator must be recognized as the key figure in controlling the plant. He must know or be able to find out any information the computer has about plant conditions or program status. He must also be able to inform the program of any data or decisions during

normal execution. During error conditions as discussed in the previous section, he may have to take control of the plant, correct the situation, and then return the plant control to the computer program. He may also have to direct the monitor to begin execution at any specified point in the network in case that in fixing the error the plant operation progresses or regresses without the knowledge of the monitor.

Because of the dynamic nature of discrete operations, an extensive operator communications system will be essential. Large amounts and varied types of information transfer can be expected, both to and from the operator. Some information, eg. error conditions will have to be relayed immediately. Control statements from the operator should also be recognized quickly. Less urgent information, eg. logs and standard progress reports as well as normal operator responses to programmed questions should not delay activity execution.

A specially designed operator's console used to enter or retrieve data would be highly beneficial. Certain functions, such as standard operator program control could be automated via a console. The console could have cathode ray tube (CRT) displays, NIXIE tubes, and status registers for fast I/O. Typewriters would be utilized for extensive data input and hardcopy output.

Another desirable feature would be to separate messages intended for the computer system personnel from process messages to the plant personnel. Separation of error messages from progress reports and data logs is also preferable.

Accepted methods for efficient I/O are buffering and asynchronous operation between the CPU and I/O devices. Buffering

involves storing the output and sending it out character by character as fast as the peripheral device can receive it under the control of an interrupt initiated driver program. Meanwhile the CPU is free to continue working (with periodic interruptions from the peripheral device indicating it is ready to accept another character from the CPU). Asynchronous operation requires hardware (data channel) which once initiated carries out the I/O operation without direct instructions from the CPU. Both operations require more complex operations in hardware and/or software.

Although buffering and asynchronous operation are desirable features, they are not always available. For instance in FORTRAN 'write' statements to the line printer on the IBM 1800 system, program execution is held up until output is completed. In order to avoid holding up execution of the monitor program with non-buffered, non-asynchronous output it is desirable to hold all non-urgent messages until activity execution has ended.

When operator response is required, rather than holding up all execution in the monitor, he should be allowed to enter the data at his convenience, including the ability to anticipate a question and enter an unsolicited response. This allows him to go about his other responsibilities using the computer as a tool rather than scheduling himself around the computer.

3.4.6 Purpose for a High Speed Monitoring Module

Under some conditions it is desirable that the main control programs of the discrete activity monitor should only be in core periodically for a set maximum time rather than be permanently core

resident. This would be the case in order to start-up a process using "background" time on a computer that is executing other priority programs, eg. DDC or in the case of processes requiring several hours for start-up during which the computer is to be available to other users.

However, in most applications there are several steps that require higher speed monitoring than could be efficiently provided by a non-core resident program. A need for a smaller core resident program with high speed monitoring capabilities is evident. Such a program could be given definite tasks such as checking various critical points in a plant for alarms, updating high resolution clocks or timers set by the main program, and limit checking of control variables. All tasks would be of short durations but require constant or rapid surveillance. Then, if a clock times out or an alarm was recognized, a service program could be called immediately into core to take care of the situation.

The high speed program would take a very small part of the total CPU time and core memory, but would allow the larger slower programs to be suspended from time to time without endangering the plant, thus freeing the computer to execute other programs.

3.4.7 Accommodation of Changing Operating Conditions and Procedures

Another unique characteristic of discrete operations is the wide variety of procedures which must be followed from time to time as the initial state of the plant and the desired final conditions are varied even slightly. For instance, there can be multiple starting (or initial) conditions for any one processing unit and each of these conditions might require a different start-up procedure. Two approaches

may be taken when such an operation exists:

1) Organize and program a separate network for each different set of procedures.

2) Define all alternatives as subsets of the most rigorous or "cold start" procedure. The program could then execute a specified subset from the one large network. Common data tables containing frequently changed parameters such as set-points, alarm limits, and control constants could then also be initialized with the proper data set before a specific run. This would alleviate the need to provide separate computer instructions with hard-coded parameters for each different procedure. A monitor system which used data easily accessed by multiple utility programs would greatly facilitate this feature.

3.4.8 Basic Operations to Provide in the Instruction Set of the Discrete Activity Monitor

The basic operations required of a discrete activity monitor can be divided into two categories. The first applies to any process control program: standard process I/O features, arithmetic and logical operations for control calculations and limit checking plus a certain amount of program communication. The second category is restricted to more discrete operations:

a) logical operations - choices as to procedure in the network.

b) wait periods - wait for a specified time or until some condition occurs.

c) extensive operator communications

d) ability to execute user written (eg. FORTRAN) programs at certain points in the network.

An option, the equivalent of "in-line coding", to allow the addition of statements from another computer language, eg. FORTRAN would add to the computing power available in the system.

Even if all these features are provided it is still important to make provision for the addition of user specified instructions particularly on the macro level.

3.5 Program Testing/Debugging - Operator Training

Prior to implementing a computer controlled application, it will be necessary to insure that there are no errors in the network logic or activity programming. In order to find some errors, it will be necessary to simulate actual operation and this should be possible without affecting normal plant operation and preferably without having the long wait periods required in real execution.

Execution in a test mode should allow the user to change the program online without altering the entire data set. Options such as table dumping and stepping through the execution would help in tracing errors in trouble spots. The ability to re-execute any portions of the network without having to execute the whole network would also save much time in debugging.

In order not to affect the process during a test run, the user's program would not be allowed to send process output or change any constants in normal programs which might be controlling the plant while a test is in progress. The test mode would be used as new procedures were adopted before actual implementation. This would permit a

gradual evolution from manual to automatic control.

All the facilities available as test options would permit and be extremely helpful in operator training. An operator could execute the program in test mode and learn the various options, features, and methods employed without endangering normal plant operation.

3.6 Project Maintenance and Improvement

The final phase to consider in the development of a project is the constant updating and improving of the overall application. Several of the procedures established in the present manual operating mode will probably be changed after experience is gained with computer control. Also, normal changes to process conditions will require updating in the programming.

Changes will be made within the program to better accommodate a specific application. Features which aid the user in evaluation of the performance of the system and his programming will be essential. In general, features which record usage of various system functions plus log actual values of variables, eg. activity durations, which are initially estimated by the user would be required. Any error conditions or uncommon operations should also be recorded for later analysis.

An outstanding feature of the network concept is that parts of the network can be debugged and implemented at a time. The application can thus be set up in modules and implemented as required rather than having to wait until the entire project is complete. In view of the large size of most applications, and the usual shortage of trained manpower, this is an extremely important feature.

The following chapter contains the description of ADAMS, a prototype program based on the design discussed in this and the preceeding chapter. ADAMS was used to evaluate some of the alternatives and was implemented to include a majority of the basic concepts considered essential in a discrete activity monitor.

CHAPTER IV

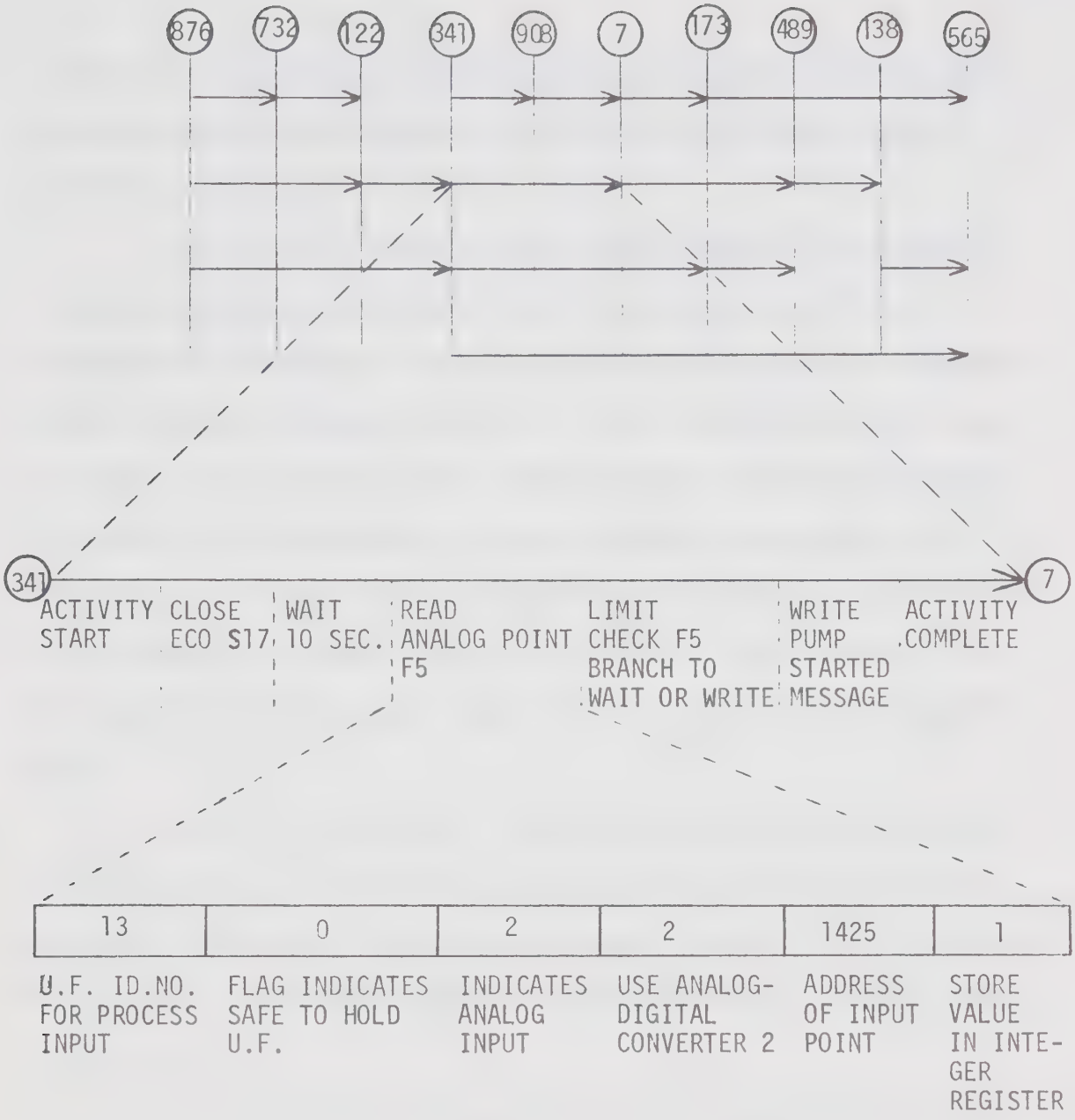
ADAMS - ALBERTA DISCRETE ACTIVITY MONITORING SYSTEM

4.1 Introduction

This chapter deals specifically with the ADAMS program, a working prototype developed as a part of the work to apply computer control to process operations consisting of discrete steps such as plant start-up. The chapter first presents an overall view of ADAMS, then deals with each general phase involved in implementing the system on some application. Project organization, detailed planning, features of the ADAMS program, testing, and project optimization are each discussed in detail in order to inform a user how to implement the ADAMS program to his application. The section on the ADAMS program is intended to give the user enough information about its functions and concepts to use it intelligently. Program details are documented in the ADAMS program user's manual[17].

4.2 Overview of ADAMS

The user's first step in implementing ADAMS is to represent his application, eg. start-up, as a network of activities where each activity performs an independent function (see Figure 4.1(a)). Next the activities are programmed using ADAMS unit functions (see Figure 4.1(b) and (c)). Unit functions are a set of routines available in the ADAMS program which permit a high-level, user oriented, programming approach. A CPM analysis calculates apriori estimates of activity latest start times which along with the network specifications and unit functions for each activity form the basic input data



a) Top: Simplified Network

b) Middle: Expansion of an Activity into U.F.

c) Bottom: Expansion of a U.F. Into a Parameter String

Figure 4.1. EXPANSION OF A PROJECT NETWORK INTO ACTIVITIES AND UNIT FUNCTIONS

files required by ADAMS. Several offline, or batch processing type, utility programs are available to facilitate data input, system initialization, etc.

The principal online or real-time programs which comprise the ADAMS system are designed to run in a timesharing or multi-programming environment. However, a small, core resident, high speed monitor performs frequent checks on critical variables in the process and updates any high resolution timers required for the application. The ADAMS real-time executive, used to schedule and execute the activities, is composed of two larger programs which are executed at a user specified interval but for not more than a set period of time, eg. they could be run every minute with a 5 second maximum execution time.

Activity scheduling is automatically handled by the ADAMS executive which, within the constraints of the network specified by the user, strives for a time optimal project duration. This relieves the user of a significant amount of programming found in some systems of this type.

In addition to the executive and high speed monitor, programs have been written to make operator communications and program control possible. This permits the user, in the initial programming, to include messages to the operator concerning progress in the network as well as asking the operator to enter information, eg. a desired setpoint, which is not available during the programming. The operator may also request program status, etc. and, if he chooses, eg. during error conditions, to halt or hold various parts of network execution.

Hence the operator can still intervene in the programmed sequence to overcome any unexpected real-time problems, a feature which is highly desirable in any process control program.

The ADAMS program also contains a test mode which permits execution of the program without affecting any part of the plant. The several options available in the test mode facilitate program debugging and operator training.

Provisions which encourage system and user program modification to facilitate project re-evaluation and optimization complete the overall project design approach considered essential in adopting new systems such as ADAMS.

4.3 Organization of the Application to be Supervised by ADAMS

Network analysis techniques are employed for project analysis, organization, and coordination. The concepts used are important not only in the organizational phase, but have been utilized to a high degree in the on-line portions of the ADAMS program.

Initially it is necessary to define exactly which steps must be taken and the order in which they must be carried out to achieve the desired end. This, considering the thousands of steps involved on a detailed level of planning plus the many interactions present a formidable managerial problem.

Network diagrams (see Figure 4.1(a)) provide a flexible, realistic, approach for modelling the application. The arrow diagram defines all the possible interactions and sequencing restrictions within the network but does not define a fixed sequence or timetable of activities. This is important since any relatively complex project

contains several orders in which it could proceed, some of which are more advantageous than others with respect to shortening overall project duration or optimizing use of physical resources.

The project is set up using increasing levels of detail. For instance consider start-up of a large chemical process. At the highest level of planning, management personnel would consider the overall problem and construct an arrow diagram illustrating the sequence in which the various component unit processes should be started, eg. unit 5A then units 5C and 6E simultaneously. These steps could then be separately sent to other groups for more detailed planning. Each group would then draw a network diagram defining what is involved at that lower level, eg. in starting unit 5A (a distillation column) one activity is to start the feed system. The activities on this level could once again be divided and a network drawn for each, eg. in starting the feed system for unit 5A an activity is to start pump P-28. Thus the activities become progressively more detailed until they can be defined by instructions to ADAMS, i.e. the unit functions. The overall network can then be assembled from its various subnetworks until the total project is represented on the most detailed level.

An important feature of this concept is that work on one level is independent of work on other levels. Using the above example, assume group 1 is assigned unit 5A and group 2 has unit 5C. A higher level of planning has decided that unit 5A is to be started first, relative to unit 5C, and this decision could be changed without affecting the planning of groups 1 or 2. Similarly group 1 might

decide to change the time at which the feed system should be started, how it is started, or how pump P-28 is started and can do this without affecting group 2 or the group planning on a higher level. Consequently, work can proceed on several levels simultaneously without conflict. Once the highest level of planning decided that unit 5A had to be started its planning could proceed, allowing work on the feed system to begin and eventually planning for the start-up of pump P-28 would begin.

The most detailed level at which an activity must be defined is dependent upon the level of ADAMS unit functions available to the user. This final level of detail defines the actual activities scheduled and executed by ADAMS. A single activity is assumed to be a string of unit functions that are to be executed sequentially although the user may initiate parallel steps provided he looks after the sequencing himself. However, as described in the example of starting three pumps simultaneously (see Section 3.4.3), it is easier to allow the system to handle parallel steps.

The final step in project organization is to estimate the duration of each activity and then perform a CPM analysis on the network. This can be done manually or by any standard CPM program which generates the estimated overall duration of the project plus the latest start times (LST) for each activity. The LST is the latest time an activity may begin without extending the estimated project duration. The LST's indicate the relative priority of uninitiated activities and are used by the ADAMS executive in an effort to achieve a time optimal overall project duration.

The organizational phase is now finished. The project is defined as a network of activities and a priority, i.e. the LST, has been obtained for each activity. The next phase will include programming each activity in terms of unit functions and entering the data into the appropriate ADAMS files.

4.4 Activity Programming

Once an activity has been defined it is programmed by selecting the appropriate series of ADAMS unit functions (see Table 4.1). Unit functions are simply separate generalized routines which make up a table driven processor used to implement a user oriented programming language within ADAMS. The user merely specifies which unit function(s) he wants in the activity and provides the proper set(s) of arguments, i.e. the unit function identification number and parameters (see Figure 4.1(c)). Hence a unit function is dependent on a particular application only in its set of user specified parameters.

Unit functions give the user low level capabilities such as basic arithmetic operations used for control calculations plus logic operations which will compare and branch to any specified unit function within the activity. Unit functions also provide higher level capabilities such as process I/O or "wait" for a given period. Also on a high level, unit functions can be used to drive other ADAMS modules, in particular for operator communications and high speed monitoring.

Although several generally applicable unit functions are included in ADAMS, it is very simple to add new ones. Hence should a

TABLE 4.1

LIST OF STANDARD UNIT FUNCTIONS

AVAILABLE IN ADAMS

U.F. ID. NO.	FUNCTION
1	START ACTIVITY
2	END ACTIVITY
3	WAIT N SECONDS (LOW RESOLUTION)
4	PERFORMS INTEGER AND REAL ADDITIONS
5	PERFORMS INTEGER AND REAL SUBTRACTIONS
6	PERFORMS INTEGER AND REAL MULTIPLICATIONS
7	PERFORMS INTEGER AND REAL DIVISIONS
8	UNCONDITIONAL OR LOGICAL SKIP ±N UNIT FUNCTIONS
9	SET UP HIGH RESOLUTION TIMERS OR REQUEST HIGH SPEED MONITORING AND LIMIT CHECKING OF SOME PROCESS POINT, OPTION TO EXECUTE A SPECIFIED ROUTINE WHEN TIMER EXPIRES OR AS RESULT OF LIMIT CHECKING
10	CHANGE OR READ ANY WORD, OR BITS IN ANY WORD, IN A SPECIFIED DDC CONTROL LOOP
11	SENDS MESSAGES AND WILL WAIT FOR REPLY FROM OPERATOR
12	PROCESS OUTPUT ANALOG OR DIGITAL
13	PROCESS INPUT ANALOG OR DIGITAL

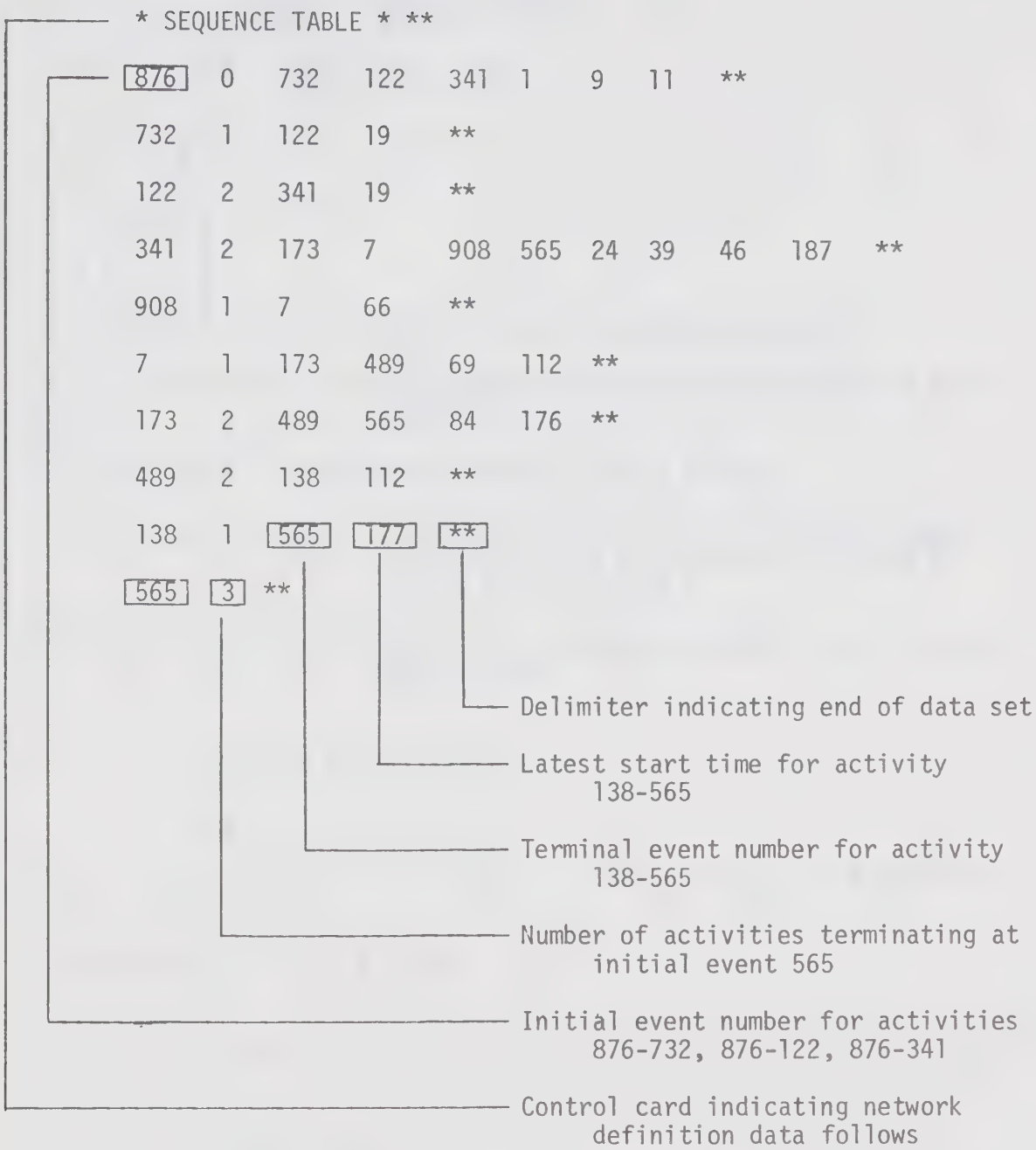
user find repeated use for a particular routine normally accomplished using several unit functions or requires a unit function specifically tailored to his application he may write and add his own unit functions. Thus both the efficiency and the capability of the ADAMS language can be increased.

After the activities are programmed the network data (initial and terminal event numbers), the CPM results (activity latest start times) plus the activity programming (unit function ID's and parameters) must be input to the appropriate ADAMS bulk storage tables. The input program used for this function reads the data from cards (see Figure 4.2) and performs error diagnostics on it before placing it in the proper file.

General error checks such as for the correct type (integer or real) and number of parameters are included in the ADAMS prototype program. Other more explicit error checks could be added by the user. Also a symbolic method of addressing widely used parameters, eg. process I/O points, logical device numbers, etc. should be added. Then if a certain parameter is changed the user would not have to go through every unit function to see which ones use that parameter.

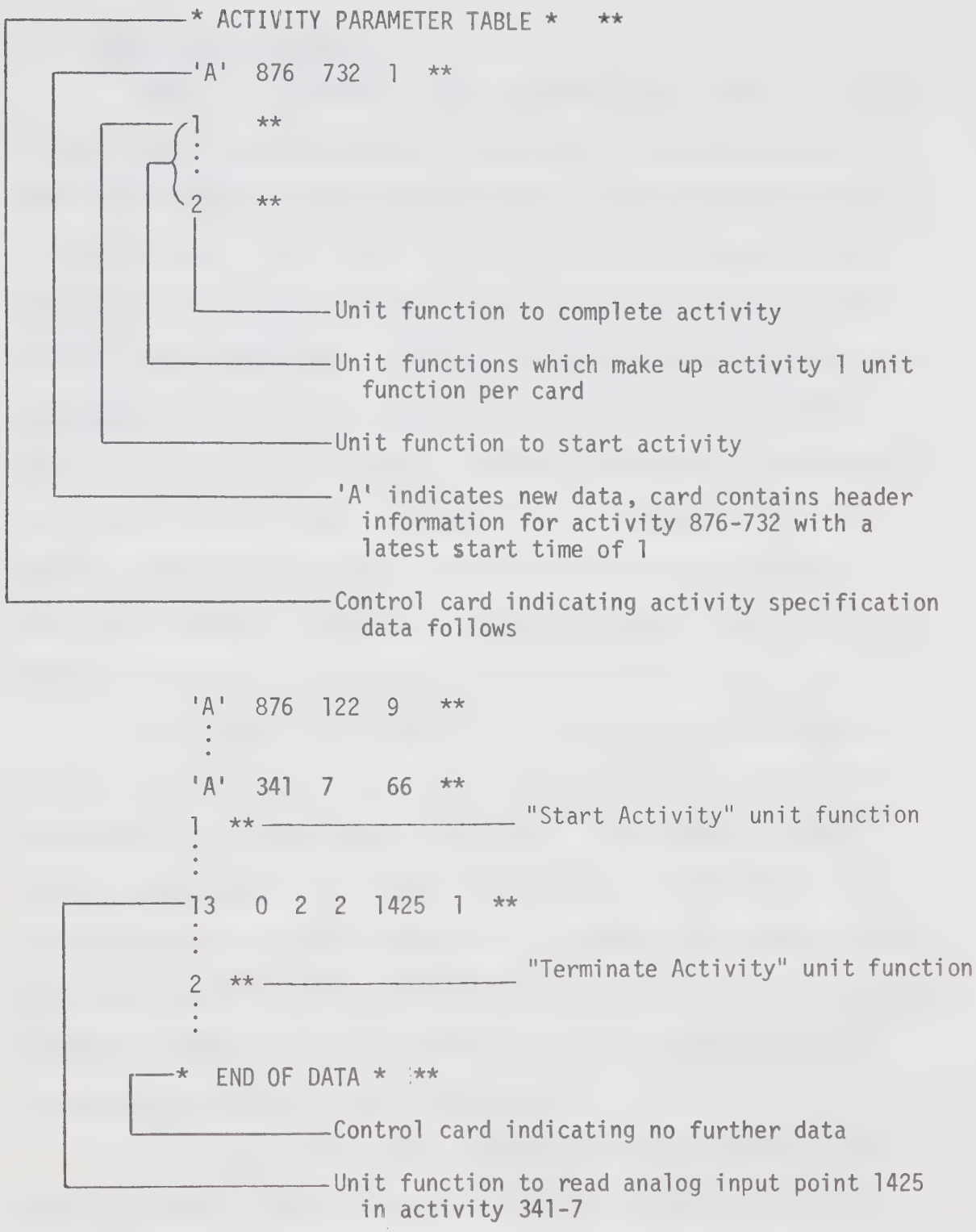
A final, as yet unmentioned type of input involves messages which are provided by the ADAMS system or are included by the programmer for operator communication during the online execution. All such messages are stored in a library and can be accessed by the operator communications module.

Having input the data, the user may now run the ADAMS program either in the test or normal operating mode.



a) Data defining network

... continued



b) Data defining which unit functions to be executed in each activity

Figure 4.2. EXAMPLE OF INPUT DATA FOR NETWORK* USED TO EXERCISE ADAMS

* network shown in Figure 4.1.

4.5 ADAMS Online Programs

ADAMS is composed of several programs (see Figure 4.3) each of which have separate functions. The heart of the system is the ADAMS real-time executive which handles the scheduling and execution of the activities. It is run at a user specified frequency for not more than a set maximum execution time. Supporting programs such as the high speed monitor which executes at a much higher frequence and the operator communications program, which is queued as required, can be requested to perform their functions from within the executive, i.e. by the unit functions. Another group of programs such as the operator communications module used for data input and programs which permit operator control over the executive are requested by the operator himself.

The ADAMS system makes use of an existing direct digital control (DDC) program at the University of Alberta for process monitoring and analog input/output operations. This program is core resident and operates at a higher interrupt level than ADAMS. In applications such as plant start-up it is assumed that control will be transferred to DDC once the discrete start-up operations are completed. The ADAMS interface to the DDC program could be easily changed to accommodate an equivalent user's program.

The various functions in ADAMS have been modularized into separate programs and have been given separate priorities and allowable execution times (see Figure 4.4) in order to run in a time-sharing or multi-programming environment which supports background work. The overall system is highly table oriented and uses in core table overlays from bulk storage files extensively. A detailed

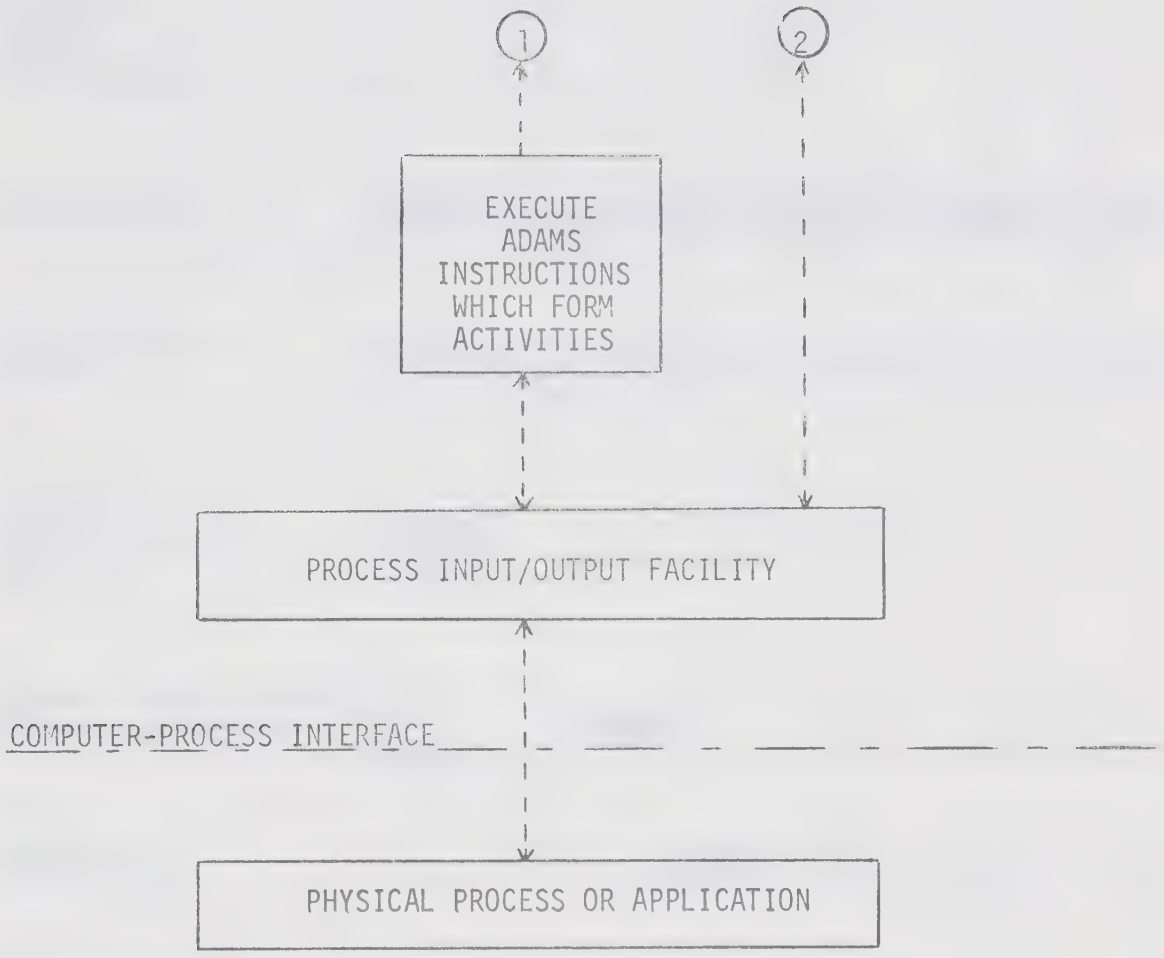
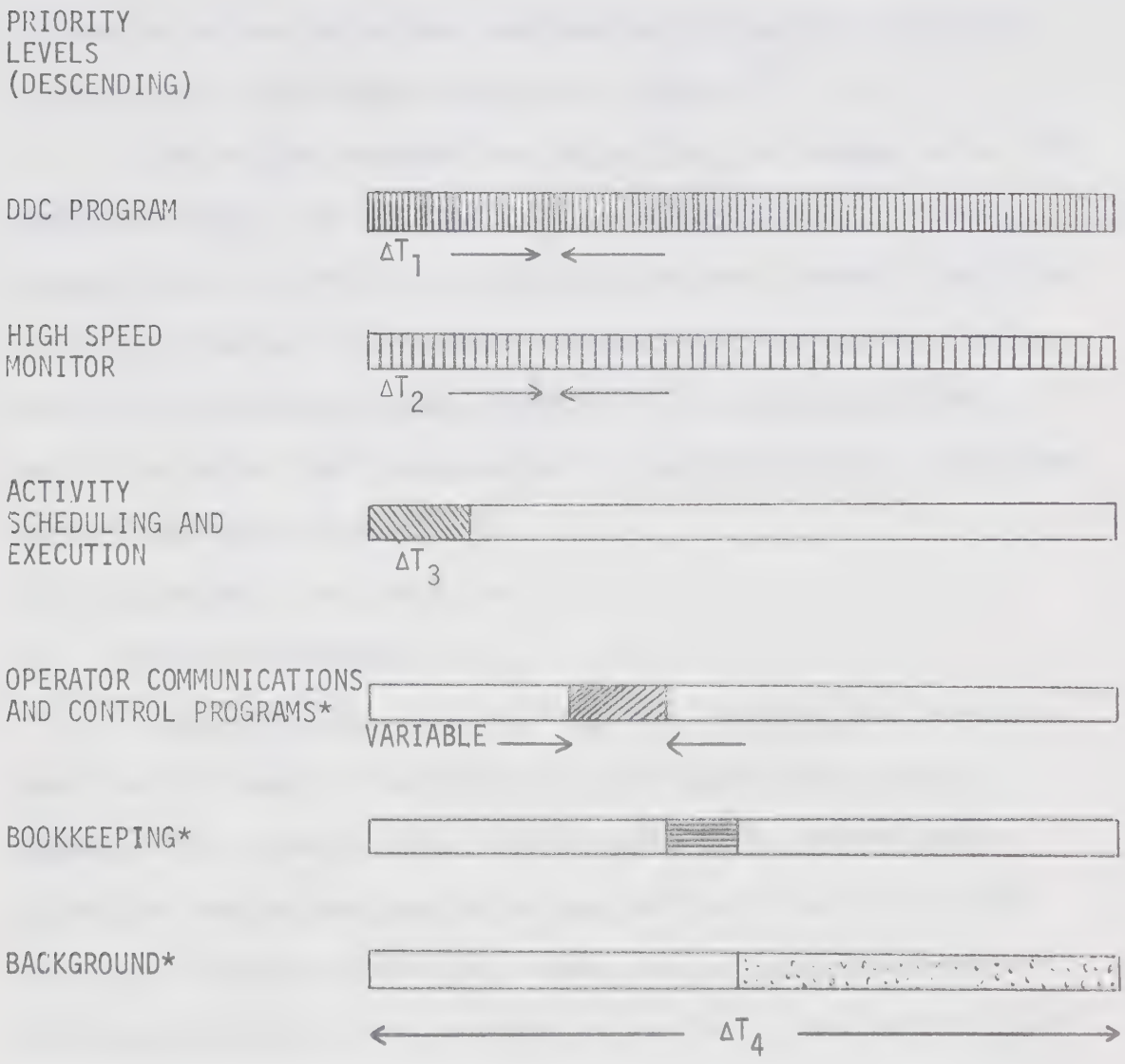


Figure 4.3. FUNCTIONAL LAYOUT OF ADAMS PROGRAMS



ΔT_1 cycle time for DDC program, eg. 1 sec.

ΔT_2 cycle time for high speed monitor, eg. 2 sec.

ΔT_3 maximum time allowed for execution of real-time supervisor, actual time taken could be less, eg. 5 sec.

ΔT_4 cycle time for activity scheduling and execution, eg. 60 sec.

* Note: under multi-programming executive system these functions can be overlapped.

Figure 4.4. ADAMS TIMESHARING STRUCTURE

explanation of the tables used and how they are updated from files is available in the ADAMS program user's manual[17].

The online execution may begin after the network data, CPM analysis results, unit function I.D.'s and parameters, and the messages for the ADAMS library have been entered. Normally the first phase would involve testing ADAMS, however, for the sake of clarity the details concerning normal operation will be discussed first. Each of the major functions provided in the ADAMS system is described in the following sections.

4.5.1 ADAMS Real-Time Executive

Activity Scheduling

The scheduling of activities (i.e. deciding when execution should be initiated) is controlled by the network data which is stored in the sequence table. The sequence table defines which activities have to complete before any particular activity may be initiated. Because of the large table size only a small portion of the sequence table is core resident at any time. The complete table in bulk storage "read only" files remains unaltered throughout the execution. The core version or segment of the table is constantly updated as progress is made through the network. The in-core sequence table is flagged and manipulated to indicate activity status:

1) activities required to have completed before initiation
2) not initiated
3) in progress
4) completed

When an activity can be initiated, i.e. when there are no

more activities to complete before it may be initiated, its name (initial and terminal event numbers) and priority (latest start time) are transferred into a buffer. This buffer may hold several activities which have been given the go ahead for execution but have not as yet been entered into the execution table. Should room not be available in the execution table the initiated activities are queued until an activity in the execution table completes. Activities are moved in order of their latest start times when such space becomes available. Hence critical activities are started while the remainder are queued.

Queuing the activities in order of their latest start times gives the executive the flexibility required in controlling a real-time process. The latest start times are interpreted as a relative criterion for starting one activity versus another. Hence if an activity in an ordinarily non-critical string of activities falls behind schedule in the real-time execution, its latest start time will indicate when it is more important to start it than an activity on a more critical path as estimated offline. This means the executive will give an ordinarily lower priority activity preference of execution if by doing so the overall project duration may be shortened.

As activities are completed their status is updated in the sequence table in core and parts of the core table are periodically overlayed by the succeeding parts from the files.

Activity Execution

Once an activity name has been transferred from the buffer into the execution table it can be processed. The table may hold

several activities at once and all will be processed concurrently. Because of the high speed of the CPU the number of activities which can be executed concurrently is usually limited by the amount of core space allocated to the table. The activities in the table are processed in order of their latest start times so the more critical activities are given more CPU time if so required.

The execution table (see Table 4.2) in core represents a small portion of the table contained in a 'read only' file. For each activity in the core table, there is a certain amount of header information such as the activity name, latest start time, flags to indicate the activity's status plus a real and an integer register which can be accessed by unit functions within that activity so that information may be passed from one unit function to another, even if they are not both in core at the same time. The remaining part of the table holds one or more unit functions for that activity.

As new activities are initiated and as the processor steps through the strings of unit functions the core table is updated by information overlays from the files. Hence the more unit functions the table can hold (a system variable restricted by core space) the less frequent the overlays. The more activities the table can hold the fewer activities have to be queued, therefore, the better the chance for shortening overall project duration. The desirable number of activities the table should hold can be determined by examining the sub-critical paths of the network. For example, if four sub-critical paths have overall durations similar to the critical path, room for five activities would be recommended.

	ACTIVITY NAME		L.S.T.	FLAGS		REGISTERS INTEGER	REAL	UNIT FUNCTION ID. AND PARAMETERS	UNIT FUNCTION ID. AND PARAMETERS
1st Activity	341	173	24	0	16	16384	0.0	. . . .	. . . .
2nd Activity	341	7	39	1	4	0	0.0	13 02 2 1425 1	3 0 10
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
nth Activity	341	565	187	4	10	-1	12.7	. . . .	. . . . .
	-	-	-	-	-	-	-	-	-

← HEADER INFORMATION →

NOTES

(1) The number of activities and the number of unit functions per activity are variables set by the user when he builds ADAMS into his system

(2) After the last unit function of an activity in the core table is executed, new unit functions are read from disk

(3) Activity execution table is restored and saved at each execution interval

TABLE 4.2

ACTIVITY EXECUTION TABLE
(CORE VERSION)

When activity execution begins (see Figure 4.5), the most critical activity is processed first. The executive executes one unit function after another within an activity until:

1) the activity is terminated,

2) a "wait" (explained below) is encountered, or

3) the time allotted for activity execution expires.

In cases (1) and (2) the execution is switched to the next highest priority activity and it is processed in a similar manner. The execution continues until all the activities have been processed and are in a suspended state or as mentioned in case (3) the allotted time expires.

The "waits" referred to above include any operation in which the CPU would be rendered inactive such as during slow I/O operations. This includes operator communications and waits within an activity (eg. wait for five minutes or wait until variable one reaches a value of N). When a unit function containing such an operation is encountered, it can initiate action and then tell the executive to begin processing the next activity. If an operation involving a wait is expected to be fairly short (eg. analog process I/O) the executive can be instructed to process the next unit function in the next activity then return to the first activity immediately. All the instructions to the executive are contained in the unit functions and require no programming by the user.

In short, activity execution has been designed to handle concurrent activities, processing them on a priority basis, and to overlap wait periods in any activity with executable operations in others. This avoids delays in execution of one activity due to a

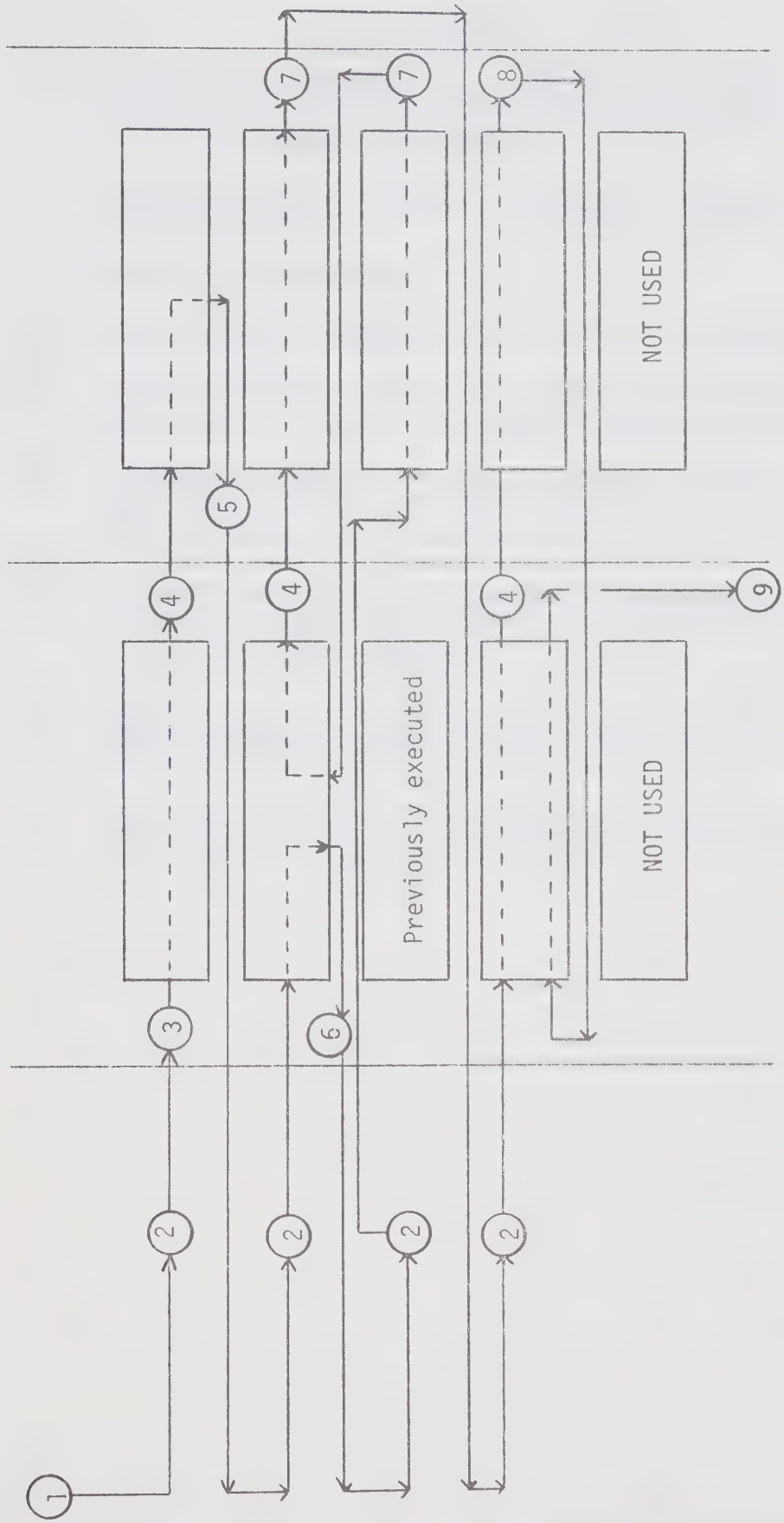


Figure 4.5. REAL-TIME EXECUTIVES PROCESSES ACTIVITIES

LEGEND

(1) Activity execution begins, executive goes to first activity in table, i.e., most critical activity as table is arranged in order of latest start times.

(2) Get pointer which indicates which unit function should be executed next.

(3) Execute unit function.

(4) Unit function completed, go to next unit function.

(5) Encountered "long wait" unit function, eg. low resolution wait N sec. or wait for operator response, go to next activity.

(6) Encountered "short wait" unit function, eg. high resolution wait N sec. or wait for signal from high speed monitor, set flag to return, go to next activity.

(7) 'Activity complete' unit function encountered, go to next activity or return to process any short wait unit function which have been encountered.

(8) Completed last unit function of activity in table, overlay space in table allocated for unit function by the next unit function and continue processing activity.

(9) Time allotted for activity execution has expired or no further unit functions remain to be processed, exit.

wait in another. Should there be no executable operations the executive will suspend itself, freeing core for other programs.

4.5.2 ADAMS High Speed Monitor

In designing the activity execution module to run at a set frequency with a maximum allowable execution time, a problem arose whenever a situation required continuous surveillance or high resolution timing. This would not be possible when the executive had been suspended for a long time, eg. fifty-five out of sixty seconds. To overcome this a separate high speed monitor (HSM) module was written. The HSM is given the highest priority in the ADAMS system, is core resident, and is typically executed every one or two seconds.

The monitor has three separate functions which can be requested by the user via the appropriate unit function:

1) update timers
2) digital input with bit checking
3) DDC status bit checking

Depending upon the result of the bit checking, or a timer timing out, the HSM can carry out any of four user selectable routines:

1) initiate digital output, eg. open a switch
2) change a word or bit in a DDC loop
   (this provides a very wide range of control options)
3) set a flag for the ADAMS executive
4) queue a specified user written program

As mentioned, options within the HSM are selected by special unit functions. For instance a user may request a timer be set up so that in 37 seconds a particular routine is executed. The unit

function gives the necessary parameters to the HSM which looks after the rest. The high speed monitor is also used for process status checking. It carries out its own digital input but relies on the DDC program to take care of the analog input and associated limit checking of the value received. These functions can be used as process alarms or as signals to ADAMS that a certain process condition has been achieved, eg. when the level reaches L_2 switch the pump off.

The HSM is composed of two modules, the first does the timer updating and process status checking, the second contains the routines and is executed only when requested by the first module. Module one would normally be core resident, executed at a one or two second frequency; module two could be kept in bulk storage if core restrictions were a problem.

The capability provided by the HSM to carry out long periods of waiting or monitoring external to the actual unit function execution means that operations initiated by unit functions can be continued:

1) when other unit functions are being processed so that concurrent activities are feasible
2) when the main ADAMS executive has suspended so that action may be taken at any time required on either a preprogrammed or "polled interrupt" basis.

4.5.3 Operator Communications

A complete and efficient communications system between the operator and ADAMS is of prime importance to the success of the system. The communications are handled largely by two separate

modules, one for operator input and one for output from ADAMS. Messages leaving ADAMS are initiated either by a unit function, i.e. the user or by the ADAMS executive to inform the operator of some change in status. The input module, which is queued by the operator, stores the information given by him in bulk storage for subsequent retrieval by the unit function.

Two types of messages are sent by the ADAMS executive. Short system error messages, composed only of a set of parameters, will be sent directly from within the executive without going through the large output module. These messages may be decoded to find the cause and location of the error as well as appropriate error recovery steps (note: such messages include "table full" warnings which indicate certain operations have to be queued until table space is available but do not require operator attention). The second type of message sent by the executive are longer and utilize the outgoing message module. Such messages may be delayed slightly since activity execution is given a higher priority. Such messages quite often are explanations of the shorter messages described above. Messages sent by unit functions are also routed through the outgoing message module.

When a message is to be sent to the operator, the unit function or the executive places certain parameters in a table located in a fixed core area. The parameters indicate the identification of the message to be sent, the debug class (explained shortly), the record number of any file containing data to be sent with the message and, in the case of a unit function, the location of an operation complete flag to indicate to the unit function when the message has

been output or when operator data entry has been completed. The outgoing message module is then queued and executes, whenever core space can be made available, using the parameters from the table. After it sends out the message, it may give an operation complete to the specified flag or if operator response is required it will set up parameters concerning the identification of the response, the logical unit it is to be entered on and the location of the operation complete flag to be set after the input is complete. The module then exits.

When the operator is ready to answer a question posed by a unit function, he queues the incoming message module. This means that at no time does a program in ADAMS demand immediate input and instead "waits" until the operator is ready with the answer. Hence the executive and other programs may continue while any number of unit functions are waiting for operator input. The operator enters the identification number and then the data requested by the unit function. The message module stores the data in a bulk storage file and flags the operation complete flag with the record number of the file in which the data is stored. The next time the executive processes the unit function, the data can be retrieved and execution can be continued in that activity.

Other features of the communication system include an option by which the operator can set flags to eliminate unwanted messages and another option allows the operator to enter unsolicited messages. As mentioned each message can be given one of ten debug levels. The operator may request that all messages on any level be cancelled. Hence a user could include many messages, especially valuable for testing or operator training, and then selectively

cancel them without re-programming his application. The levels may be turned off and on during real-time execution so should trouble crop up during execution the operator could allow all the diagnostic messages to be sent which would normally be filtered out. The option of entering unsolicited responses allows the operator to anticipate a routine request and answer it before it is asked. This would allow the operator more freedom to then go about his other tasks plus saving time in the activity execution.

In summary, a message, with or without a request for operator response, will not hold up execution of other activities nor will it hold up execution of other programs. Also the process operator may enter data at his convenience rather than have to respond immediately to requests from the computer.

4.5.4 ADAMS Modes of Operation

In order to safeguard plant operation in times of problems, eg. equipment failure, three alternatives were considered:

1) suspend or "hold" ADAMS execution
2) restart execution at a safe place in the network
3) shutdown the operation

These options permit the plant operation to be suspended temporarily to recover from error conditions, then restart the execution or else shutdown the plant.

Hold

The hold mode is implemented so that control of the application may be withdrawn from ADAMS in whole or part and turned over to the operator. Three types of holds are available to the operator:

1) hold initiation of events

2) hold initiation of activities

3) hold execution of activities

The three holds are designed to give the operator maximum control over and flexibility in the execution.

An event is a node in the network diagram which marks a point where one or more activities terminate and one or more activities may begin. The initiation of an event occurs after all activities which terminate at that event have completed. The placing of a hold on the initiation of an event means that none of the activities which begin at that event are allowed to be put into the execution table. Hence a hold on the initiation of events allows all activities which have been initiated to complete, but will not permit new activities to start. This option permits the operator to implement any network as a series of separate steps.

The second type of hold, an initiation of activities, has much the same effect as a hold on initiation of events but it is more selective. Holding initiation of activities means that any activity which can be started as a result of another activity(s) completing is not transferred into the execution table. This does not restrict the execution of any activities which have already started. This hold is more selective in that individual activities may be held whereas for the hold on events several activities could be held when one event is held. Both holds can be used to freeze parts of or the entire network at a prespecified location. This would be helpful to the operator who, after several activities had completed, wanted to hold further action until he checked out the plant, or correct some instrument failure, before allowing execution to resume.

The third hold, on execution of activities, is more useful for correcting problems which have occurred within a particular activity. A hold on activity execution halts the execution of unit functions in the activity at the first "safe" location encountered, hence this hold actually affects the unit functions. The user has the option, during activity programming, to set a "not safe to hold" flag in the unit functions. For instance, in starting a reactor it might be essential to continue executing the unit functions once a certain point is reached. The executive will not hold unit function until it has checked to make sure it is in a safe condition. A hold on execution of activity(s) can be initiated either by the operator or from within a unit function which has recognized an error (eg. unit function to change a word in control loop 122 of the DDC program cannot find the loop). If the unit function requests a hold, it is placed on a particular activity regardless of the "not safe to hold" flag since continuing might worsen an error condition.

The ability to place any of the discussed holds on the network, release the holds, and request the present status of the operation with respect to holds is provided for the operator in a separate operator queued program. Each of the holds can be put on in three degrees:

1) to affect the overall network
2) to affect the overall network excepting specified locations
3) to affect only specified locations.

This gives the flexibility required when only parts of the network should be held. For example, an operator upon discovering an error

could hold all activity execution. Then, as he has time to determine the extent of the error or as the plant recovers he can release certain specified activities from the hold.

Restart and Shutdown

ADAMS has no provision as yet for automatically restarting the activity execution in specified locations of the network. This is considered a necessary function and should be implemented during further development of ADAMS. At present, the shutdown is envisaged as a separate network which would be executed if recovery from an error condition was not considered feasible. With the present ADAMS program the user must program his own logical checks on the plant operations and specify transfers to specific unit functions or to a hold mode.

4.6 Testing/Debugging Options in ADAMS

Upon completion of data entry it is possible that errors might still exist in the organization of the project and in programming activities. Several options are available for use under a "test" mode of execution to find and correct these errors prior to actual implementation. These options permit the user to omit any "wait" instructions and to execute ADAMS at computer speed and without affecting plant operations. Also of importance, the test mode is designed to facilitate operator training, i.e. familiarize him with the application under ADAMS computer control.

The basic test options include:

1) divert process output, record on peripheral device.
2) skip over wait periods.

3) divert any changes to DDC loops.

4) step through activities executing one unit function per activity during each scan of the real-time executive.

5) online table dumping.

6) online changes to activity unit function parameter table (future work).

The test mode can also initiate holds which is particularly helpful if the operator wishes to follow the execution of one part of the network at a time.

The test mode is very flexible in that an operator may use all or any combination of the options. For example, it would be possible for ADAMS to follow a manual operation by diverting any action which affects the process but allowing monitoring of the process and input of information as usual.

Because it is possible to implement chosen sections of an overall network for computer control, testing is important to debug new sections while other sections are already running under computer control.

4.7 Project Maintenance and Optimization

A large number of changes in operating procedures and conditions can be expected of any real-time process.

In the application of ADAMS more reliable data will be available for planners and programmers after executing the project several times. For example, with better activity duration estimates more meaningful CPM analysis results can be obtained and the

scheduling of activities by the real-time executive will have more effect. As the project becomes more stable and reliable under ADAMS supervision, control limits can be optimized and the overall ADAMS structure and timing can be tailored to the specific application of the user.

Recording of several key variables can be logged by peripheral programs:

1) activity duration
2) operator interventions
3) system errors
4) various modes of operation used
5) equipment failures

Several of the concepts and features employed in ADAMS facilitates rather than discourages making changes. As mentioned the network concept separates the scheduling from the programming. This means details can be changed easily without changing the network. Changes to the programming consist of changing routine parameters which can be done without recompleting or re-assembling the program. Changes to the ADAMS system have also been considered. The table driven approach, modular construction, and user specified program priorities and timing assist the user in modifying ADAMS to fit his application.

4.8 Summary

Among related systems, ADAMS is unique in that it is designed around and incorporates network analysis concepts. This provides a realistic well defined framework around which the user may

develop his project. The organization of his personnel and project planning can be divided into independent subsystems for detailed work, then combined into the overall project. Network analysis is also utilized in the scheduling and execution of activities by the ADAMS real-time executive. This feature combined with the ability of ADAMS to supervise concurrent activities is also unique. A priority system, based on CPM analysis, insures that jobs which will eventually determine the project duration are carried out as soon as possible. The scheduling and execution of several concurrent activities is automatically handled within ADAMS. The user, after drawing an arrow diagram for the project, has only to program the functions within each activity; activity scheduling, a large task normally programmed by the user, is automatically controlled.

Activity programming is accomplished using a table driven processor provided within ADAMS. The unit functions or routines available in the processor are process control oriented and designed for use by any project engineer. Common process operations and system concepts such as servicing interrupts are automatically looked after within the easy to use high level unit functions. The table driven processor concept permits features such as the following to be easily added:

1) online editing
2) addition of user written instructions
3) self documenting input processor
4) thorough error diagnostics
5) symbolic referencing.

ADAMS is designed to be general and easily implemented on

any user's system. The use of FORTRAN and techniques to accomplish system functions such as queuing of operator I/O make ADAMS independent of any particular system (although less efficient with respect to core storage and execution time). Modular construction allows the user to tailor ADAMS to his computer system and his application. This also allows the user to rewrite any function in ADAMS to make it more efficient (eg. rewrite into machine language) without affecting the remainder of the system. For example, although it has presently run under a timesharing and a multi-partion executive at the University of Alberta, it could be easily modified to run on a dedicated computer.

Overall, ADAMS presents a logical pattern for the user to follow. Well defined stages for project development, programming, testing and debugging online execution and the inevitable project optimization and maintenance have been individually considered and provided. ADAMS represents an attempt to assist the user with all the major steps in an application and is not restricted to assistance with the computer aspects.

CHAPTER V

PRACTICAL APPLICATION OF ADAMS: COMPUTER CONTROLLED START-UP OF A PILOT PLANT DOUBLE EFFECT EVAPORATOR

5.1 Introduction

The ADAMS programs and design concepts were successfully tested by simulation and by application to the start-up of a pilot plant double effect evaporator. ADAMS was first implemented on an IBM 1800 computer using the TSX operating system. For the simulation tests the 1800 was interfaced to an analog computer on which a model had been set up to simulate some aspects of a real process. For example, there was analog I/O between the model and the 1800 which represented process signals or measurements and control signals. Two DDC control loops were implemented to control variables in the model by manipulating other variables. The functions or options available in ADAMS were extensively exercised using a network designed especially for this purpose.

Subsequently ADAMS was converted to run under the IBM 1800 MPX operating system. Under TSX all the ADAMS programs ran in the variable core area (V-core) which had 9.7 K 16 bit words. The entire core size was 32 K, the remainder of which was occupied by the executive and other system programs such as the DDC program. The machine size was expanded to 48 K and the MPX system adopted. Under MPX, the core is partitioned into several areas which are different sizes and normally assigned different priorities for the programs executing in them. Extensive use is made of disk bulk storage and the core swapping ability of MPX.

ADAMS is presently composed of 12 coreloads, the largest of which are the two which form the ADAMS real-time executive. Each of these require approximately 9.7 K words of core memory which includes the FORTRAN I/O package; 8.5 K is suffice if the FORTRAN I/O is included in the systems program area. The ADAMS programs reside in approximately 50 K words of disk bulk storage. File space, a variable dependent upon the application for the evaporator start-up, requires approximately 25 K words. During execution the real-time executive was limited to twenty seconds of CPU time in any forty-five second period and the high speed monitor was executed every second. These figures, also dependent on the application, were chosen using the evaporator start-up as an example.

In order to demonstrate its capabilities when applied to a real process, ADAMS was used to start up a pilot plant double effect evaporator. This application is the subject of the balance of this chapter.

5.2 Evaporator Description

The pilot scale evaporator (see Figure 5.1) concentrates a blended, preheated, water-triethylene glycol feed stream from about .032 weight fraction to a final product normally about .101 weight fraction. The evaporator consists of two effects which are connected in a forward feed scheme, the product from the first effect becoming the feed for the second effect which produces the final product. The heat required to concentrate the glycol solution is provided by high pressure steam introduced into the first effect.

The first effect of the evaporator is a calandria type in which the boiling fluid circulates naturally by convection. The second

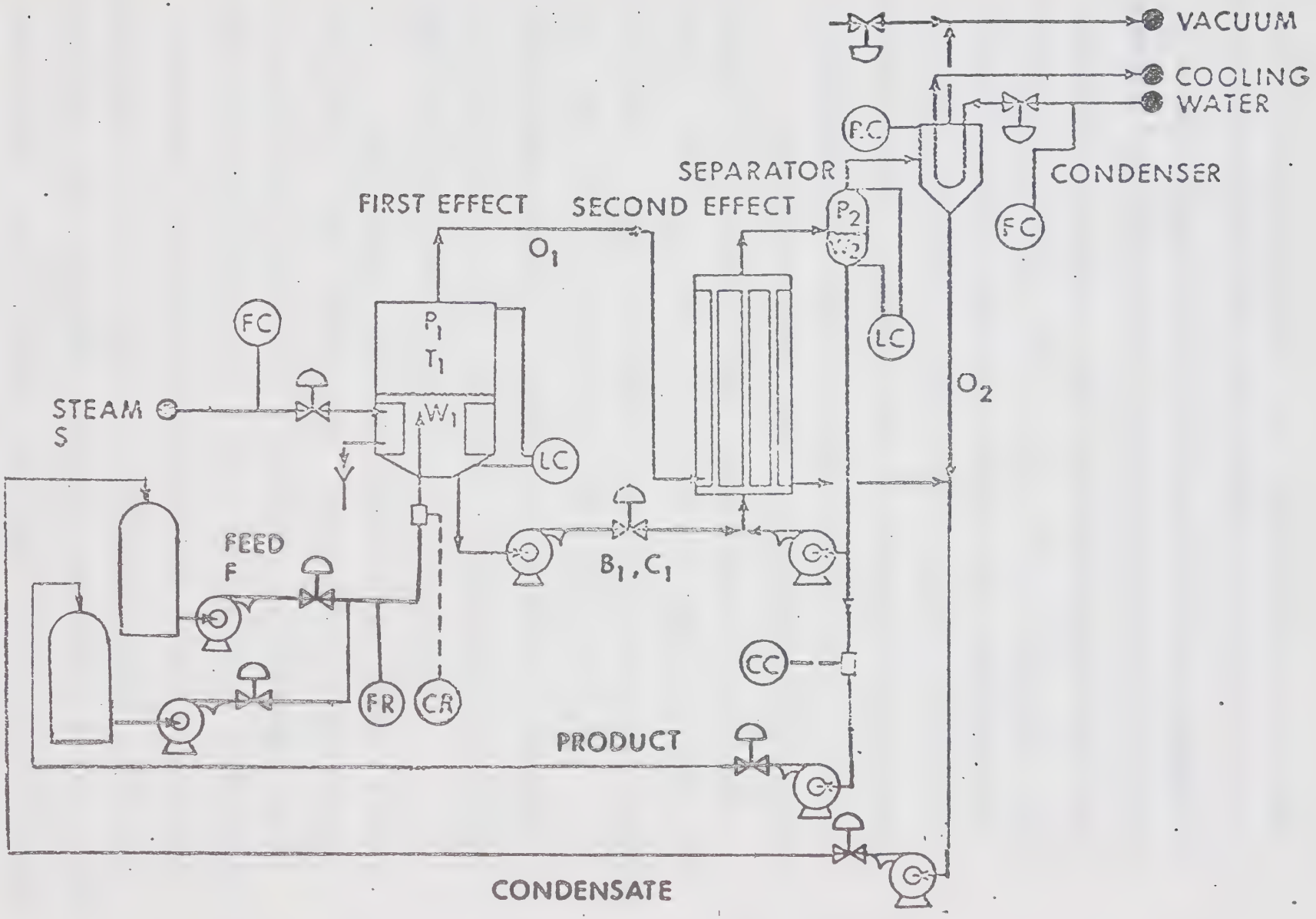


Figure 5.1 PILOT PLANT DOUBLE EFFECT EVAPORATOR

effect is a long tube vertical type in which the fluid is circulated by an external pump. Other major components of the plant include the feed (product) storage and blending system, the cyclone separator and condenser. The feed or product storage consists of four tanks, three of which hold concentrated glycol solution and one which holds water. The concentrated product from the second effect is cooled and returned to the solution feed tanks and the condensate from the condenser and second effect steam chest is returned to the water feed tank. The feed distribution and product recovery configurations can be modified by repositioning the solenoid and hand valves.

The two feed streams, water and concentrated glycol solution, are pumped from the tanks through steam preheaters and then blended into one stream which enters the first effect. Heat is supplied to the first effect by steam which enters the steam chest from a building supply line. After the solution leaves the first effect (typically a 5 - 10 minute residence time) it is piped to the second effect. The vapour from the boiling solution in the first effect is piped overhead to the second effect steam chest where it serves as the heating medium.

The solution in the second effect is recirculated by a pump through three, 6 foot, vertical tubes. Heat transfer is made possible by a vacuum which is maintained in the condenser and separator and has the effect of lowering the boiling point of the solution. Heat transfer is also aided by the forced recirculation of the solution through the heating tubes. The recirculating solution passes through a cyclone separator which assists the vapour in escaping from the boiling fluid. The vapour is drawn into the condenser, condensed, and returned to the evaporator feed tank. Concentrated product is drawn off the recirculating

stream, cooled, and pumped back to the concentrated solution tanks. Standard operating values are outlined in Table 5.1.

The evaporator is fully instrumented with two concentration analyzers, three level readings, two pressure readings, nine flowrate measurements and thirty-seven temperature readings. It has a complete set of conventional controllers and is also fully interfaced to the IBM 1800 computer for data acquisition and control functions. The interface includes analog input of 11 key measurements, analog output to position eleven control valves, and thirty-two digital output points which provide on-off control for the solenoid valves and pump motors.

5.3 Evaporator Start-Up

5.3.1 Procedure

Introduction

The evaporator start-up procedure discussed below was developed to demonstrate the overall capability of ADAMS. Hence, as many as possible of the sequencing and programming options of ADAMS have been included and the demonstration is designed to make the separate steps evident to an observer. Several features of the ADAMS programs are demonstrated in the start-up:

1) Real-time activity scheduling by the executive
2) Queuing of the least critical activities when there is not enough core to process all activities until core becomes available
3) Ability to execute activities concurrently
4) Use of the high speed monitor
5) Operator communications system

TABLE 5.1

STANDARD OPERATING VALUES

Variable Description	Value
Steam flow	2.00 lb./min
Total feed flow	5.00 lb./min
First effect bottom flow	3.49 lb./min
First effect overhead flow	1.51 lb./min
Circulation rate	190.00 lb./min
Product flow	1.58 lb./min
Second effect overhead (separator) flow	1.91 lb./min
Feed concentration	.032 wt. fraction
Product concentration	.101 wt. fraction
First effect level	16.00 in.
Second effect (separator) level	11.00 in.
First effect pressure	6.0 psig
Second effect (condenser) pressure	-17.0 Hg
Total feed temperature	190 degrees F.
Steam temperature to first effect	277 degrees F.
Steam temperature to second effect	223.6 degrees F.

6) Communication with other communication programs
   eg. the DDC program
7) Intermixing of manual and computer automated operations
8) System initiated holds
9) Use of non-interruptable strings of unit functions during critical operations which override normal timing of the executive.

This start-up procedure assumes that the evaporator is to be run in the normal forward-feed operating configuration. The control system used during start-up (see Figure 5.2) manipulates the rate of feed entering the first effect to quickly establish and maintain the desired liquid level. The inter-effect flow is manipulated in a similar manner to control the second effect level and the final product rate is fixed. Once start-up has been accomplished, the control system is reconfigurated by ADAMS to a scheme that is more suitable for continuous operation (see Figure 5.2). The operator role in the start-up is to carry out operations which are not handled automatically and to provide judgement as to process stability.

Outline of Procedure

The computer first ensures that the evaporator is set up properly for an automated start-up. The operator is asked to manually shut off the solenoid valves and then the computer opens the control relays using digital output (the manual switches and the computer driven relays are in parallel circuits so both must be open to turn the circuit off). The operator is then requested to open the hand valves on the utilities and feed tanks and to set the control panel to permit computer control of the control valves.

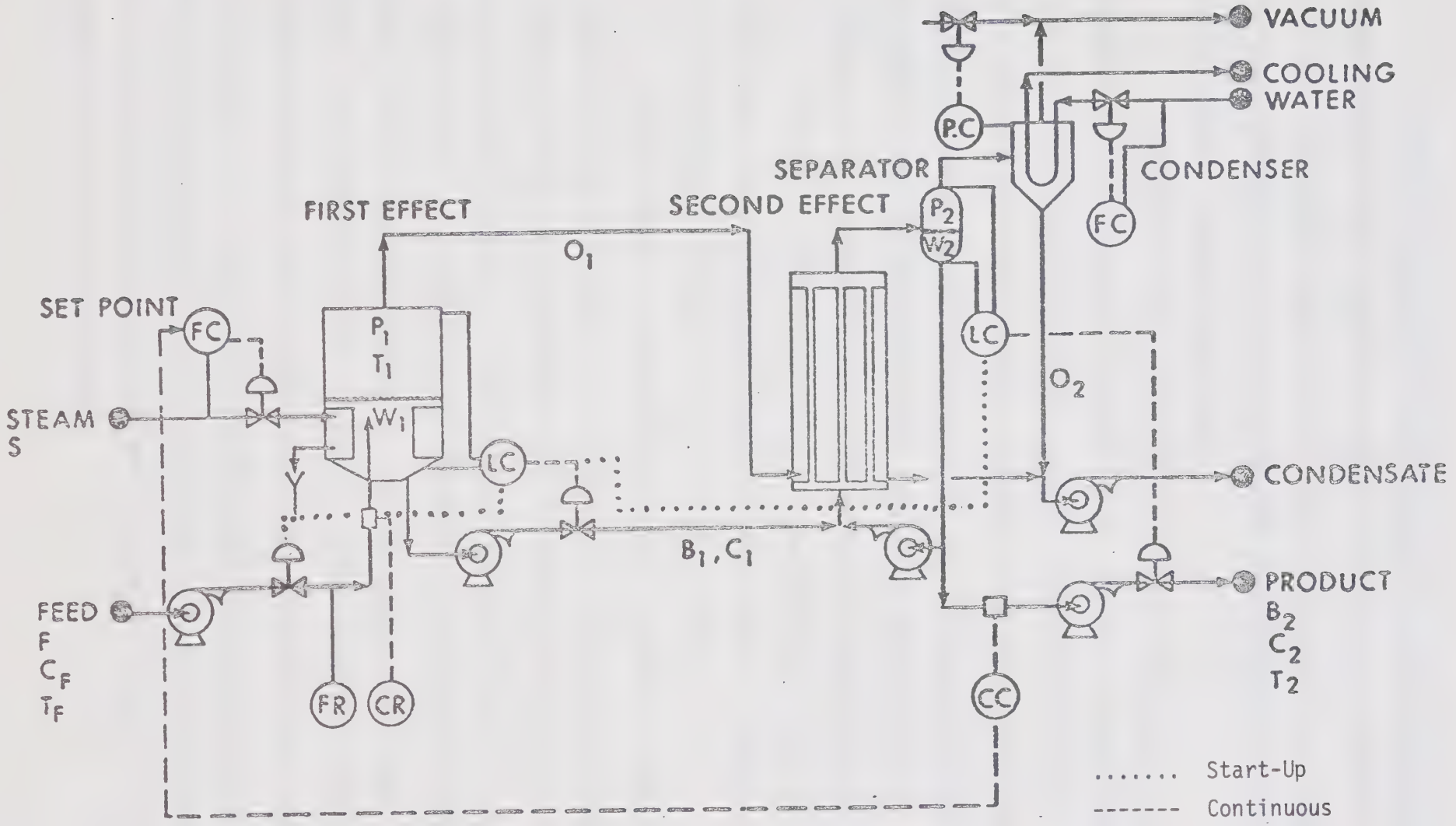


Figure 5.2 CONTROL CONFIGURATIONS USED ON EVAPORATOR

After the standard DDC table has been entered by the operator, ADAMS makes the loops inoperable-manual, and configurates them for start-up. The current output stations, which control the positioning of the control valves, are switched to the auto position by the operator and the computer closes all the valves.

The procedure now ensures that steam and cooling water are supplied to the evaporator. The operator is informed if proper flow-rates cannot be established so that he may investigate and correct the situation before the start-up is resumed. The pumps on the feed streams are started automatically and the flows are sensed to ensure the pumps are working satisfactorily. Once again the operator is informed of any discrepancies and is requested to investigate the cause.

At this point the operator is requested to open purge vents on the steam chests of the two effects and live steam is introduced to heat the equipment and purge any inerts from the chests.

As the steam is introduced, the level in the first effect is established by DDC control of the feed flow. The operator is given a choice of using the inter-effect pump or operating using the natural pressure differential as a driving force between the effects. If he requests the use of the pump it will be started by the computer and checked for satisfactory operation in the same manner as the feed pumps. After the level in the 1st effect reaches a specified minimum value the 2nd effect level is placed under DDC using the inter-effect flow to control it. After the 2nd effect level builds up sufficiently, the recirculation pump is started, the product flowrate placed under DDC at a fixed setpoint, and the calibration pump is turned on to return condensate to the water feed tanks.

After the time specified for steam purging has expired the operator is requested to close the vent valves and the input steam flow is put under DDC.

Next the vacuum utility is checked and the vacuum system and the differential vapour pressure loop are placed under DDC.

At this point all the systems within the evaporator are running in the start-up mode with tight level control. It is possible to run indefinitely in this manner, however, further programming is provided to return the evaporator control systems to its normal operating mode. This is done in two steps. First, the operator is requested to inform ADAMS when the evaporator has reached steady state. At this time the control scheme is changed so that the feed rate is fixed, the first effect level is controlled by manipulating inter-effect flow, and the second effect level is controlled by manipulating the product flowrate. The second step is carried out after the operator again informs ADAMS that steady state is reached. The normal integral controller constants are entered into the loops and the feed preheater system is started under DDC.

The final step in the start-up involves opening the computer controlled relays on the solenoid to permit manual manipulation.

Explanation of the Activities

The functions of each activity within the network (see Figure 5.3) are more fully described below.

Activity 1 - 10. The ADAMS operator communications system is used to inform the operator that ADAMS has begun the evaporator start-up.

Activity 10 - 20. The operator is requested to enter the DDC table for evaporator control and acknowledge when it is in. The system will halt further execution of this activity until the operator responds but will continue processing any other parallel activities (in this case Activity 10 - 30). When the table is in, all the loops are made non-operable and manual and a message is sent out informing the operator of the present status of the loops. This ensures that if the loops were already present, they are initialized to an inactive condition.

Activity 10 - 30. This activity contains the instructions to the operator which must be carried out manually in order to prepare the evaporator for start-up. The system waits until the operator acknowledges that the steps are complete. Basically this involves opening manual valves on the utilities to the evaporator and setting the feed and product recovery systems to the desired configuration. Also the control panel must be switched to the DDC mode to permit computer control of the automatic valves. Normally the evaporator is in the correct configuration so the operator will just enter that manual preparation is complete. At this point the system continues and turns off all computer driven relays for the various solenoids on the evaporator and then turns on the main power supply to the evaporator. These operations are carried out by the "process output" unit function using the digital output option.

Activity 20 - 30. This is a dummy activity to ensure that both activities 10 - 20 and 10 - 30 have completed before activity 30 - 50 is allowed to start. A dummy activity is used only to show sequencing dependencies and takes zero time. It is composed of a start activity and a stop activity unit function.

Activity 20 - 40. The two DDC loops which control the levels in the first and second effects are reconfigurated for start-up. Extensive use is made of the change loop option of the DDC communication unit function. The main changes made to each of the loops are to lower the scan rate from thirty-two to sixteen seconds, to change manipulated variables to the start-up configuration, to enter control constants for tight level control (ie. a high proportional constant with no integral action), and to change flags which specify what action is taken when the high or low limits on error, input, or output of the DDC loops are exceeded. The last change is made in view of the severe control action which could be taken with the high proportional control constants. Examples of changes to whole words and to individual bits within the DDC table are contained in this activity. When the changes are complete a message informing the operator of the new loop status is sent.

Activity 30 - 50. A message requesting that the operator verify that the current output stations are in the auto mode is issued. The current output stations drive the I/P transducers which in turn drive the control valves to the desired positions. Upon operator verification, the unit function for process output, in this case DAC pulsed anolog output, is used to zero the current to all control valves. The same unit function is then used to open the solenoid (digital output) which allows instrument air to the controller and valves. The operator is informed of the action.

Activity 40 - 50. Dummy activity (see activity 20 - 30).

Activity 50 - 60, 50 - 70. These are used to ensure that steam and cooling water are available to the evaporator. The activities are similar and hence will be discussed together. The activities use a variety of unit functions, including accessing and changing the DDC table, digital output, utilization of some of the mathematical options, waiting in the high speed monitor, branching between unit functions within an activity, and operator communications. The activities also include the option which overrides the normal execution of the real-time executive (ie. running at a set frequency for a maximum interval in core). This safety feature holds the executive in core to process an activity which is considered to be in too hazardous a position to be left unattended. This option is invoked through one of the unit function parameters.

The procedure used in both activities is to make the controlling DDC loop operable, open the solenoid valve on the supply line, open the control valve on the supply line and using the reading obtained from the DDC loop decide whether there is sufficient flow. After the control valve is opened a short wait is implemented using the high speed monitor unit function. The flowrate is then obtained from the DDC loop and compared to a fixed value. If it is not sufficient the procedure is repeated up to three times. The counter for the number of times the action has been retried is maintained by the programmer in one of the registers provided for each activity. If a satisfactory flow has not been established after the three attempts the control valve is closed and a message is sent to the operator informing him of the difficulty and requesting his intervention. The time between the opening and closing of the control valve is considered hazardous and hence all the

unit functions involved in this part of the activity are flagged "unsafe to leave unattended". This forces the executive to keep processing the activity until the valve is closed. After the operator checks out the situation he may request the system to retry the procedure or he may tell it that everything is functional and the procedure may be bypassed. If the flaw is detected satisfactorily the control valve is closed and a message stating so is sent.

Activity 50 - 80, 50 - 90. These are very similar to activities 50 - 60 and 50 - 70 except they are used to determine whether the feed water and feed solution pumps have been started successfully. One minor difference is that in addition to opening the feed solenoids and bypass solenoids on the feed system the pump is also started by the computer using digital output. After the pump is started the procedure for detecting the feed flows is identical to that used in detecting the steam and cooling water flow.

Activity 60 - 70, 70 - 80, 80 - 90. These are dummy activities.

Activity 90 - 100. This activity is used to warm the evaporator equipment and purge any inerts from the steam chests by injecting steam slowly for a couple of minutes with the purge valves in an open position. Since the purge valves are manually controlled operator confirmation that they are open is requested and then the steam control valve is opened using pulsed DAC output to the current output stations (COS). The operator is then told he may place the COS on manual and close the control valve and purge valve whenever he feels the operation is complete. The DDC loops controll-

ing the condensate levels are then made operable and automatic to control the steam condensate levels in the steam chests. Cooling water flow through the condenser is also initiated.

Activity 90 - 110. Liquid levels in the first and second effects are established and the remaining pumps are started. The procedure used is to "fill" the evaporator using the feed flow as a manipulated variable. The first effect level control loop and its slave are made operable and automatic. While the level is being established the operator is asked whether or not the inter-effect pump will be used and to set the hand valves around the pump to the proper configuration. When the operator has responded and the level in the first effect has built up sufficiently the second effect level control is started using the inter-effect flow as a manipulated variable. The inter-effect pump (if required) is also started by a computer actuated relay. The operator is informed as the levels are put under DDC. When the second effect level is considered satisfactory the recirculation pump and the calibration pump are started automatically with the product flow at a fixed rate.

Activity 100 - 130. The steam and feed water flowrates are controlled at fixed setpoints.

Activity 110 - 120. The vacuum utility is connected to the evaporator. The same procedure employed in activities 50 - 60 and 60 - 70 is used in this activity.

Activity 120 - 130. The vacuum system is placed under DDC and brought to the normal operating level. The differential vapour pressure control of the vent from the second effect steam chest is also started. After

waiting briefly to allow a vacuum to be established in the system the second effect vent solenoid is opened. This purges the vapour spaces of any inerts and thereby improves heat transfer. The high speed monitor is used to time the duration which the solenoid is left open and then to close it automatically.

Activity 130 - 140. All systems (excepting the feed preheat) in the evaporator have now been started under the start-up control scheme. The evaporator is quite safe running in this mode and may be left so indefinitely by halting the execution of ADAMS at this point. If the operator wishes he may allow execution to continue and the evaporator will be brought to the normal control scheme. This is done in two phases, the first being the change from using the flows into the effects to control the levels to using the flows exiting the effects to control the levels. This is done after the operator has acknowledged that the process is in a stable condition and the change can be made safely. Once this change is made, the operator again is requested to indicate when the process has stabilized. The normal control constants for proportional plus integral action are then entered in place of the constants for tight level control. The feed preheater system is also placed under DDC and the start-up is complete.

Critical Path Analysis

The final stage of planning is the estimation of the activity durations and the subsequent CPM analysis. Table 5.2 contains the estimates of activity durations and the corresponding activity latest start times. These have been obtained through a

TABLE 5.2

ACTIVITY DURATION AND LATEST START TIMES

Activity Names		Durations ($\frac{1}{1000}$th HR)	LST
1	10	1	0
10	20	58	6
10	30	63	1
20	30	0	64
20	40	12	83
30	50	31	64
40	50	0	95
50	60	17	95
50	70	15	97
50	80	16	96
50	90	14	98
60	70	0	112
70	80	0	112
80	90	0	112
90	100	103	141
90	110	107	112
100	130	7	244
110	120	10	219
120	130	22	229
130	140	291	251

few trial runs but could change as more reliable data is obtained from the start-up.

5.3.2 Programming the Start-Up

Several of the general features of ADAMS discussed in previous chapters, are clearly illustrated by this application.

The most obvious is the ease with which a few unit functions can be used to accomplish a relatively complex task, i.e. that the high level capabilities of ADAMS unit functions are extremely powerful.

The desirability of being able to implement "macro" type procedures (unit functions) has also become apparent. This is particularly noticeable in starting pumps, an operation which is repeated five times in this application. The addition of a user written "start pump" macro type procedure for the general situation would be highly advantageous. Unit functions which would work from a list are also recommended. For example, in activity 10 - 20, all the DDC loops are made non-operable and manual, each step requiring a separate unit function. One unit function which would execute a given list would be more efficient, both from a programming standpoint and during execution.

Several activities demonstrate the ease of intermixirg manual operator functions with computer controlled functions. This capability is necessary in applications where automation progresses from a partially manual to a more automated state, or when it is not feasible to allow computer control of some functions. This is the case in activity 10 - 30 during which the operator must position certain hand valves and ensure the control panel is set up for computer control. Most of these tasks could be automated by installation of appropriate hardware and

computer interfaces. However, complete automation would be difficult to justify economically.

The usefulness of the "not safe to hold" flags which can be set in the unit function programming is obvious in activities such as 50 - 60. In this case the steam valve is opened to test whether there is flow into the evaporator. It is important that this valve be closed before the executive suspends operation since this could leave the process in an unsafe condition.

In general, experience with unit functions has proven them to be easily and effectively used.

5.3.3 Testing/Debugging

The test mode has been used extensively and is particularly helpful in debugging the unit function programming while the evaporator is operating normally. The most useful features in such a case are the "trap process output" and "cancel changes to the DDC table" options. For example in activities 10 - 20 and 10 - 30 the computer makes the control loops for the evaporator inoperable and manual and shuts off all the solenoids, pump motors and the main power supply. Normal operation would have to be suspended if these test options were not available in ADAMS.

The ability to bypass waits is also a desirable feature during testing. Waits for a specified period of time may be ignored to speed testing.

Another option used extensively during testing was to step through an activity executing one unit function per scan. This was useful in activity 10 - 30 when several solenoid valves are turned off. Each operation could be checked out to make sure the correct action was obtained.

An option, particularly useful in conjunction with stepping through activities, is the ability to dump system tables which indicate how the system is working. The operator may select any output device he wishes to receive the dumps, thus not interfering with the operator's I/O device. Examination of these tables by a system analyst could point out several types of difficulties the system might be having with an application.

It was found that the hold options were used extensively during testing, both to isolate complicated areas within the network for inspection using the table dump options or to slow the network execution to the point where an operator could follow it closely for training purposes. An example of both these cases begins at event 50 which initiates the activities which start the feed streams, test for steam, and test for cooling water. During normal execution the four activities occur almost simultaneously making it very difficult to follow individual steps. By initiating an overall hold and then releasing it activity by activity the operator can follow the execution closely.

The test mode was found to be very flexible in that all or a combination of test options can be put in effect at any one time.

5.3.4 Implementation

At present the ADAMS programs have been written and debugged on both TSX and MPX systems.

The example of the evaporator start-up as just described was programmed, successfully tested and debugged, and can now be used at any time.

5.3.5 Improvement of the Start-Up

As with any new project, it is expected that several changes will be made to the planning, programming, etc. of the start-up procedure as experience is gained. ADAMS has been specifically designed to facilitate such changes.

Activity duration estimates (see Table 5.2) are expected to be altered. This will mean the real time executive will become more effective in scheduling the activities as more reliable durations are used in the CPM analysis. This will be particularly felt in cases where activities have to be queued due to a lack of core table area required for activity processing.

Unit function parameters are also expected to change. For instance activity 120 - 130 purges the inerts from the evaporator through the vacuum system for a set period of time. This time can easily be changed if so desired.

The start-up procedure itself might be changed. For example activity 130 - 140 contains steps which allow steam to enter the feed preheaters. This is presently one of the final steps in the start-up. It would be possible and perhaps more efficient to allow this step to occur as steam is introduced to the first effect. This could easily be done by removing the pertinent unit functions from activity 130 - 140 and placing them in activity 100 - 130.

The systems approach which is the basis of ADAMS permits a variety of changes to be made easily while affecting the rest of the application to a minimum degree.

5.4 Experience with the System

The application of ADAMS as a working system to the start-up of the evaporator took place over a span of approximately one year. During this time the author was employed at an industrial computer installation and therefore was able to compare ADAMS to commercially developed systems. It is felt that ADAMS has a sound design basis which offers significant advantages over other approaches. The standard of programming, testing and debugging aids, and documentation, also compare favorably with industrial systems.

The use of ADAMS to achieve a computer controlled plant start-up has proven a successful project. The usefulness of computer control in this application is particularly obvious in assisting new students who are unfamiliar with the evaporator pilot plant. The natural pitfalls of computer control such as not being programmed for real time failures and removal of normal operator judgement can easily be circumvented by ADAMS since it allows intermixing of manual and automatic operations as well as suspension of automated procedures when necessary. During the past year of actual operation ADAMS has proven to be safe, efficient, rapid, and repeatable. Naturally, with experience, several steps and safety checks were added to the start-up procedure. These changes were incorporated smoothly, with only isolated parts of the overall procedure being affected, as they would have to be for success in a commercial environment.

Various features of the original design of ADAMS such as more extensive input checking and data manipulation facilities and a process operator's console have been deliberately omitted from the existing programs due to time limitations but are recommended for industrial use.

5.5 Conclusion

Based on the use of ADAMS during its development, i.e. the tests using the analog model and the implementation of the evaporator start-up it is concluded that

1) The network approach is well defined and easy to use.
2) Unit functions are effective and relieve much of the programming load.
3) The ADAMS programs work as designed.
4) ADAMS has been proven suitable for meaningful applications and has great potential for commercial applications.

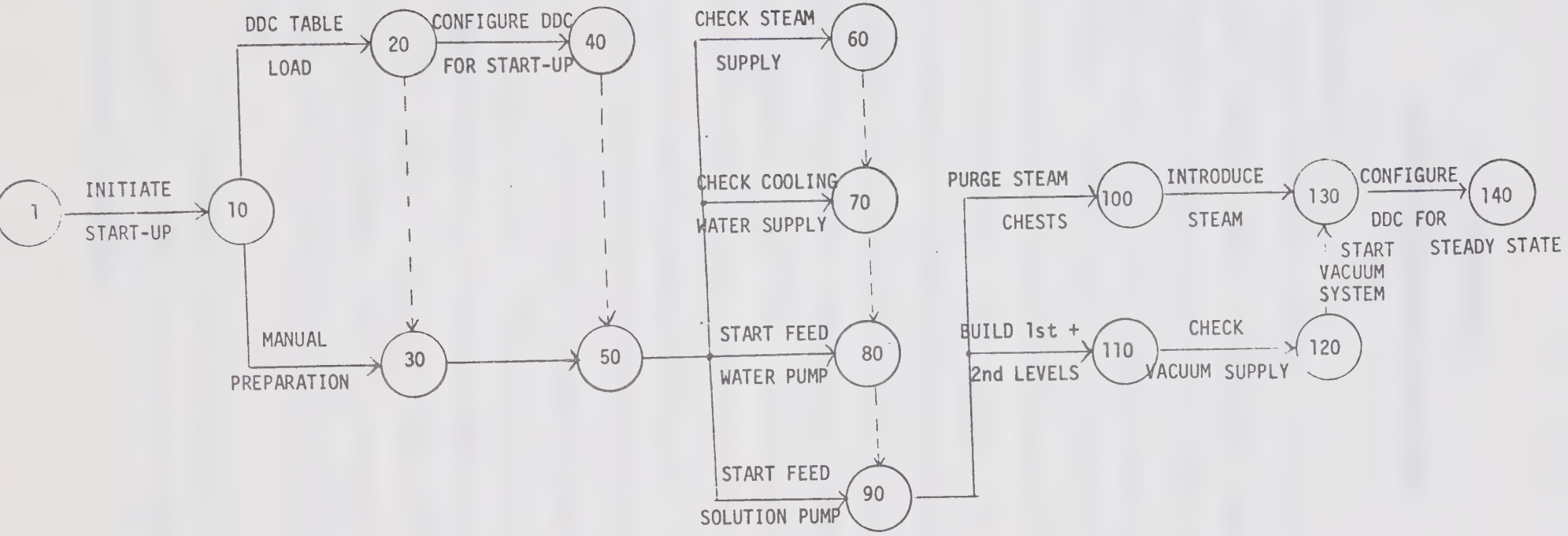


FIGURE 5.3 NETWORK DIAGRAM FOR EVAPORATOR START-UP

CHAPTER VI

RECOMMENDATIONS AND CONCLUSIONS

6.1 Recommendations

The objective of this work was to design and implement a system to apply computer control to operations composed of discrete steps. The prototype program based on the ADAMS design has established the feasibility of several useful concepts such as the use of network diagrams. However, practical experience has also confirmed the need for several of the design specifications discussed in this thesis but not yet implemented in the program. It is felt that the success of ADAMS thus far warrants additional work to the existing program in order to make it more flexible with respect to recovering from process problems and more adaptable to a process's changing operating procedures and conditions. The present system should also be improved with respect to user-system interaction, i.e., improved communication and system documentation features. More specifically, the major areas for future development are considered to be:

1) Installation of an ADAMS operator's console which would include digital readout, CRT displays, status switches, and commonly requested options on individual demand switches.

2) Utility programs to implement features such as: editing the activity parameter table as it resides in storage, printing formatted process logs on demand, and compiling records of system usage.

3) An operator control program to re-initialize and restart execution of a network at any desired location to aid in recovering from error conditions.

4) Changes to the real-time executive to allow the execution of subsets of a network. This would provide the flexibility required to handle situations such as the "cold" and "hot" initial conditions during start-up applications or the possibility of several batches being run on the same or different reactors requiring slightly different operating procedures. This flexibility would permit the user to represent several conditions as subsets of the most complex network rather than programming each of them independently. This feature could also be used in conjunction with point 3 to aid process restarts.

5) The establishment of a common data area which would contain frequently changed parameters used by the unit functions. Hence, rather than having to modify a parameter within every unit function that uses it the particular parameter could be stored in the common table and changed only once.

6) When users, who are not familiar with ADAMS, start implementing new applications a data input processor will become essential. Since it is the interface between the applications engineer and the ADAMS programs any improvements to the processor will be of direct benefit to the user. Refinements would include its own CPM analysis program, specific error diagnostics for each unit function and symbolic referencing.

7) Existing unit functions in ADAMS can be either "permanently" core resident or stored in bulk storage and "loaded on call". An extension to the input processor could be made so that the user has the ability to generate "in-line coding", i.e., his coding would be compiled, stored and executed in the same manner as

a regular unit function.

Several other minor recommendations plus a more detailed discussion of the implementation of the above points are included in the ADAMS program user's manual[17].

6.2 Conclusions

1) ADAMS, which has been successfully used to start up a pilot scale double effect evaporator, has demonstrated that the application of computers to the control discrete operations is feasible and practical.

2) Use of network analysis techniques offers advantages: in the implementation of an executive which handles the scheduling and execution of steps on a real-time basis; in that it permits separation of the project planning from the detailed activity specification or coding; and because it reduces the amount of programming required of the user.

3) The most important design specifications for a discrete activity monitor system include: a systematic approach for the planning and coordination of large projects both on the management level and on the detailed level required for execution on a computer system; an online executive which can cope with unexpected occurrences without endangering the operation; extensive provision for operator intervention and communication; features which permit thorough offline testing of the user's application and facilitate modification to to accommodate changing operating conditions and procedures.

4) The feasibility of a monitor, generalized with respect to the computer system on which it is run and to the application which it controls has been successfully demonstrated by the

existing ADAMS project.

5) ADAMS represents a starting point for the development of a commercial system for industrial use.

BIBLIOGRAPHY

1. "DDC Manual", Department of Chemical and Petroleum Engineering, University of Alberta, (1969).

2. "IBM 1800 Process Supervisory Program PROSPRO 1800", Users' manual H20-0474-1, Application Program H20-0261-0, Language Specifications manual H20-0473-1,(1967, 1968).

3. "BICEPS Supervisory Control Program", General Electric Co. Bulletin GET 3559,(1969).

4. Battersby, A., "Network Analysis for Planning and Scheduling", 2nd Ed., Macmillan and Co. Ltd., London,(1967).

5. "IBM 1800 Multiprogramming Executive Operating System Programmer's Guide" GC 26-3720-3,(1970).

6. Itahara, S., "Direct Digital Control for Batch Processes", Chem. Eng., 75, 25, 195, (1968).

7. Bacher, S. and Kaufman, A., "Computer-Controlled Batch Chemical Reactions", Ind. Eng. Chem. 62, 1, 53, (1970).

8. Carlo-Stella, G.R., and Johnson, J.A., "Sequential Logic Simulation for Industrial Computer Control", Instrum. Technol., 16, 12, 60,(1969).

9. Baldridge, B.H., Balls, B.W., and Thoraldsen, N.F., "Automatic Start-Up - A Step Toward the Unattended Plant", I. Chem. E., Symposium Series, 24, 151, (1967).

10. Whitman, K., "Starting a Pump Under Computer Control", Chem. Eng., 74, 1, 81, (1967).

11. Baker, W.J. and Weber, J.C., "Direct Digital Control of Batch Processes Pays Off", Chem. Eng., 76, 24, 121, (1969).

12. Mecklenburgh, J.C. and May, P.A., "Protran, a Fortran Based Computer Language for Process Control" Automatica, 6, 565, (1970).

13. Wade, A.I.H. and Noble, J.S., "PROSEL A New Process Control Language", Control and Instrumentation, 2, 2, 30, (1970).

14. Garzanelli, J.W., "Batch A Process-Oriented Language for Batch Sequence Control", Instrument Society of America, 22nd Annual ISA Conference and Exhibit, Sept. 11 - 14, Chicago, Illinois, Preprint No. D1-2-DACHCOD-67, (1967).

15. Ewing, R.W., Glahn, G.L., Lackins, R.P. and Zartman, W.N., "Generalized Process Control", Chemical Engineering Progress, 63, 1, 104, (1967).

16. "A Proven System for Direct Control of Batch Processes", Foxboro Bulletin L28.

17. Hayward, S.W., "ADAMS Program User's Manual", Department of Chemical and Petroleum Engineering, University of Alberta, (1971).